Sound Security Protocol Transformations

Author(s):
Nguyen, Binh Thanh; Sprenger, Christoph

Publication Date:
2013

Permanent Link:
https://doi.org/10.3929/ethz-a-007611582

Rights / License:
In Copyright - Non-Commercial Use Permitted
Sound Security Protocol Transformations

Binh Thanh Nguyen and Christoph Sprenger

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

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

1 Introduction

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

Protocol transformations are interesting for at least two applications: we can use them (1) to develop (families of) protocols by refinement [16,9,15,7,6,4] and (2) to abstract existing protocols for the more efficient tool-based verification of their properties [11]. Abstraction and refinement correspond bottom-up and top-down views on (the same) protocol transformations. To be useful, protocol transformations must be sound with respect to a relevant class of security properties, i.e., refinement must be property-preserving, or equivalently, abstraction must be attack-preserving.

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

⋆ This work is partially supported by the EU FP7-ICT-2009.1.4 Project No. 256980, NESSoS: Network of Excellence on Engineering Secure Future Internet Software Services and Systems.
transformations with respect to an expressive property specification language based on [14].

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

The appendix contains the proofs that are missing in the main text (Appendix A to D) and the treatment of structural transformations (Appendix E).

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

\begin{align*}
K4(1). \quad & A \to S : A, B, n_A \\
K4(2). \quad & S \to A : \langle B, t_S, n_A, k_{AB}, A, t_S, k_{AB} \rangle_{sh(B,S)}, \langle A, t_A \rangle_{k_{AB}} \\
K4(3). \quad & A \to B : \langle A, t_S, k_{AB} \rangle_{sh(B,S)}, \langle c, t_A \rangle_{k_{AB}} \\
K4(4). \quad & B \to A : \langle t_A \rangle_{k_{AB}}
\end{align*}

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

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

\begin{align*}
K5(2). \quad & S \to A : \langle B, t_S, n_A, k_{AB} \rangle_{sh(A,S)}, \langle A, t_S, k_{AB} \rangle_{sh(B,S)}
\end{align*}

In the second step, we eliminate the forwarding of $B$’s ticket by $A$ by applying structural transformations. This yields protocol K3, on which we verify mutual
authentication of $A$ and $B$ (P3). We omit the message $K_3(1)$ which equals $K_5(1)$.

\begin{align*}
K_3(2) & : S \rightarrow A : \{B, t_{SA}, n_A, k_{AB}\}_{sh(A,S)} \\
K_3(3) & : S \rightarrow B : \{A, t_{SB}, k_{AB}\}_{sh(B,S)} \\
K_3(4) & : A \rightarrow B : \{c, t_A\}_{k_{AB}} \\
K_3(5) & : B \rightarrow A : \{t_A\}_{k_{AB}}
\end{align*}

In the third step, we remove the key confirmation phase, i.e., messages $K_3(4)$ and $K_3(5)$. For the resulting protocol, $K_2$, which we omit here, we verify the authentication property (P2).

In a final transformation, we remove the server timestamp $t_S$ and the initiator nonce $n_A$. The result is protocol $K_1$ for which we verify secrecy (P1).

\begin{align*}
K_1(1) & : A \rightarrow S : A, B \\
K_1(2) & : S \rightarrow A : \{B, k_{AB}\}_{sh(A,S)} \\
K_1(3) & : S \rightarrow B : \{A, k_{AB}\}_{sh(B,S)}
\end{align*}

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

We can also view these transformations in the other direction, as a development of $K_4$ by refinement. We start from the abstract protocol $K_1$ satisfying session key secrecy (P1) and add new properties or modify the protocol structure with each refinement step. We verify properties (P2) and (P3) for $K_2$ and $K_3$, respectively, knowing that they are preserved by further refinements. By refining given protocols in different ways, we can develop entire protocol families, whose members share structure and properties. For example, most server-based key establishment protocols can be derived from $K_1$.

## 2 Security protocol model

### 2.1 Term algebra

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

\[
\mathcal{T} ::= V \mid U \mid F \mid C \mid pk(U) \mid pri(U) \mid sh(U, U) \mid h(T) \mid \langle T, T \rangle \mid \{T\}_T
\]

We use $\mathcal{T}$ as a shorthand for $\mathcal{T}(V, U, F, C)$ in the generic case where the concrete parameters do not matter. We denote by $|t|$ the size of a term $t$. The set $St(t)$ denotes the set of subterms of $t$. For $T \subseteq \mathcal{T}$, $vars(T)$ and $atoms(T)$ denote the sets of variables and atoms in $St(T)$. A term without variables is called ground.
The terms pk(A), pri(A), and sh(A, B) for A, B ∈ U denote the public key of A, the private key of A, and a symmetric key shared by A and B. We define the function (·)−1 on ground terms t as follows: pk(A)−1 = pri(A), pri(A)−1 = pk(A), and t−1 = t otherwise. Next, we define a number of functions on terms in \( T \).

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

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

\[
\text{split}(u) = \begin{cases} 
\{u\} & \text{if } u \text{ is not a pair} \\
\text{split}(\langle u_1, u_2 \rangle) = \text{split}(u_1) \cup \text{split}(u_2) & \text{otherwise.}
\end{cases}
\]

We also define \( \text{split}(U) = \bigcup_{u \in U} \text{split}(u) \) for a set \( U \) of terms.

**Definition 2.** We define the set \( \text{acc}(t) \) of accessible subterms of a term \( t \) by

\[
\begin{align*}
\text{acc}(u) &= \{u\} & \text{if } u \text{ is a variable, atom, or hash} \\
\text{acc}(\langle u_1, u_2 \rangle) &= \text{acc}(u_1) \cup \text{acc}(u_2) \\
\text{acc}(\{u\}_k) &= \text{acc}(u)
\end{align*}
\]

### 2.2 Protocols

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

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

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

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

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

Given a protocol \( P \), let \( V_P, R_P, F_P, \) and \( C_P \) be the sets of variables, role names, fresh values, and constants appearing in the roles of \( P \) (i.e., \( R_P = \text{dom}(P) \)). We assume a constant \( \text{nil} \in C \setminus C_P \) and define \( C_P^{\text{nil}} = C_P \cup \{\text{nil}\} \). We denote by \( \text{Event}_P \) the set of all events in the protocol \( P \) and \( R_{t_P} = \text{term(\text{Event}_P)} \). Moreover, we define the set of protocol messages (over the atomic messages of the protocol \( P \)) by \( M_P = T(V_P, R_P, F_P, C_P^{\text{nil}}) \).
2.3 Attacker model and operational semantics

Let $A$ and $TID$ be infinite sets of agents and thread identifiers. We partition $A$ into non-empty sets of honest and compromised agents: $A = A_H \cup A_C$.

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

\[
\begin{align*}
    inst_{tid}(w) & = w^{tid} & \text{if } w \in \mathcal{V} \cup \mathcal{F} \cup \mathcal{R} \\
    inst_{tid}(c) & = c & \text{if } c \in \mathcal{C} \\
    inst_{tid}(k(R)) & = k(R^{tid}) & \text{if } R \in \mathcal{R}, k \in \{pk, pri\} \\
    inst_{tid}(sh(R, S)) & = sh(R^{tid}, S^{tid}) & \text{if } R, S \in \mathcal{R}
\end{align*}
\]

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

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

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

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

The trace $tr$ as well as the executing role instance are symbolic (with terms in $M^\sharp_P$). 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., $IK(tr) = \{ t \mid \exists i. (i, \text{snd}(t)) \in tr \}$. 

---

**Fig. 1.** Intruder deduction rules
\[
\begin{align*}
\text{SEND} & \quad \text{RECV} \\
\text{SEND} & \quad \text{RECV}
\end{align*}
\]

Fig. 2. Operational semantics

We associate with each protocol \( P \) a fixed ground initial knowledge \( IK_0 \) and assume that \( C \cup A \cup F^* \subseteq IK_0 \). In particular, \( \text{nil} \in IK_0 \).

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

\[
\forall i \in \text{dom}(th). \exists R \in R_P. \text{th}(i) = (R, \text{snd}(t) \cdot tl) \land \sigma(R) \subseteq A,
\]

where \( \text{inst}_i \) is applied to all terms in the respective protocol role.

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

2.4 Type system

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

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

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

The abbreviation $\mathcal{Y}_P = T(V_p, \{\alpha\}, \{\beta_f | f \in F_p\}, \{\gamma_c | c \in C^\text{nil}_p\})$, defines the set of types of a protocol $P$. We derive the canonical typing environment $\Gamma_P : V_p \rightarrow \mathcal{Y}_P$ for the protocol $P$ from the honest substitution $\delta_P$ as $\Gamma_P = \{(X, \tau) | X \in \text{dom}(\delta_P) \land \emptyset \vdash X\delta_P : \tau\}$. Note that $\Gamma_P$ ranges over ground types.

**Proposition 1 (Type inference).** Let $P$ be a protocol and $t \in M^2_P$. Then there is unique ground type $\tau \in \mathcal{Y}_P$ such that $\Gamma_P \vdash t : \tau$.

By this proposition, we can extend $\Gamma_P$ to all terms $t \in M^2_P$, i.e., we have $\Gamma_P(t) = \tau$ if and only if $\Gamma_P \vdash t : \tau$. We say that a substitution is well-typed if the terms in its range respect the types of the variables in its domain.

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

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

3 Protocol transformations

Following Hui and Lowe [11], we distinguish two kinds of protocol transformations: message-based and structural transformations. Message-based transformations are functions on protocol messages, which we lift to events and protocol roles. In contrast, structural transformations apply directly to protocol roles.
We cover essentially the same structural transformations for splitting and relay-\[11\]ing as the splitting transformation splits selected events with pairs into two events and the relaying transformation removes a \(\text{rcv}(X)\) and a subsequent \(\text{snd}(X)\) event from a protocol role. In Section 1, they together justify the step from protocol \(K5\) to \(K3\). The other abstractions, from \(K4\) to \(K5\), from \(K3\) to \(K2\), and from \(K2\) to \(K1\) are obtained by message-based transformations. Here, we mainly focus on message-based protocol transformations. Structural transformations are discussed in Appendix E.

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

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

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

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

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

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

\[
t\theta = u\theta \Rightarrow f(t) f(\theta) = f(u) f(\theta)
\]  
(1)

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

\[
T\theta \cup IK_0 \vdash u\theta \Rightarrow f(T) f(\theta) \cup f(IK_0) \vdash f(u) f(\theta)
\]  
(2)

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

\[
f(t\theta) = f(t) f(\theta).
\]  
(3)

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

\(^{1}\) For now, you can read \(f(\theta)\) as the composed substitution \(f \circ \theta\).
We solve this problem by typing variables and restricting $\theta$ to well-typed substitutions. In Section 3.3, we thus introduce a restricted class of type-based message transformations, where a message’s type uniquely determines how it is transformed. We use the type system from Section 2.4, which enables a fine-grained control over the transformations. In Section 3.4, we show that the substitution property (3) holds for type-based transformations $f$ and well-typed substitutions. Then we lift these transformations to protocols and establish well-definedness and the simulation property. Section 4 treats attack preservation.

### 3.1 Message transformations

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

**Definition 5 (Normalization).**

\[
\begin{align*}
\text{nff}(t) = t & \quad \text{if } t \text{ is a variable or an atom} \\
\text{nff}(h(t)) = \text{nff}(t) & \quad \text{if } \text{nff}(t) = \text{nil} \text{ then else } h(\text{nff}(t)) \\
\text{nff}((t_1, t_2)) = \text{nff}(t_1) & \quad \text{if } \text{nff}(t_1) = \text{nil} \text{ then nil else } \text{nff}(t_2) \\
& \quad \text{else if } \text{nff}(t_2) = \text{nil} \text{ then } \text{nff}(t_1) \\
& \quad \text{else } \text{nff}((\text{nff}(t_1), \text{nff}(t_2))) \\
\text{nff}(\{t\})_u = \text{nff}(t) & \quad \text{if } \text{nff}(t) = \text{nil} \text{ then nil else } \text{nff}(t) \\
& \quad \text{else } \{\text{nff}(t)\}_{\text{nff}(u)}
\end{align*}
\]

We say that a term $t$ is in normal form iff $\text{nff}(t) = t$.

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

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

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

where $\mathcal{P}(u, t) = \text{split}(t) \subseteq \text{split}(u) \wedge \text{set}(\text{split}(t)) = \text{set}(\text{split}(u))$ and $\{m\}_k$ denotes the $a$-fold encryption of message $m$ with the key $k$. 


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

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

### 3.2 Deducibility preservation

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

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

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

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

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

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

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

Definition 8. \( f : \mathcal{T} \rightarrow \mathcal{T} \) be a message transformation on \( \mathcal{T} \) and \( \theta : \mathcal{V} \rightarrow \mathcal{T} \). Then we define the substitution \( f(\theta) = \{ (x, f(\theta(x))) \mid x \in \text{dom}(\theta) \land f(x) = x \} \). Note that \( \text{dom}(f(\theta)) \subseteq \text{dom}(\theta) \) as \( f \) may map some variables in \( \text{dom}(\theta) \) to nil.

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

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

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

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

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

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

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

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

We introduce a simple generic form of recursive function specifications. Based on these we will then define type-based transformations. Below, we use typing environments of the form \( \Gamma : V_{pt} \rightarrow Y_P \) with pattern variables rather than message variables in the domain. Otherwise, the type system remains the same.

**Definition 11.** Let \( f \) be an unary function symbol. A function specification for \( f \) with respect to a typing environment \( \Gamma : V_{pt} \rightarrow Y_P \) is a list of equations
\[
E_f = \left[ f(p_1 : \pi_1) = u_1, \ldots, f(p_n : \pi_n) = u_n \right],
\]
where each \( p_i \in P \) is a linear term pattern and \( \pi_i \in Y_P \) is a type pattern such that \( \Gamma \vdash p_i : \pi_i \). The \( u_i \) are terms, built from the pattern variables in \( p_i \), cryptographic operations, and the function symbol \( f \).

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

**Definition 12.** A set of type patterns \( S \subseteq Y_P \) is complete w.r.t. a set of ground types \( T \) if, for all \( \tau \in T \), there is \( \pi \in S \) such that \( \tau = \pi \theta \) for some \( \theta : V_{ty} \rightarrow Y_P \).

**Example 2.** We define \( E_0(f) \), the “homomorphic” function specification for \( f \) with respect to a typing environment \( \Gamma_0 : V_{pt} \rightarrow Y_P \) below. Clearly, any set of patterns including the set \( \{ h(X), \langle X \rangle, \langle X, X' \rangle \} \) is complete with respect to composed ground types in \( Y_P \).

\[
E_0(f) = \left[ f(h(X) : h(X)) = h(f(X)), \quad f(\langle X \rangle : \langle X \rangle, \langle X, X' \rangle) = \langle f(X), f(X') \rangle \right]
\]

Transformation specifications

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

**Definition 13 (Type-based message transformation).** A type-based message transformation for a splitting protocol \( P \) and function symbol \( f \) is a triple \( S_f = (T_f, \Gamma_f, E_f) \) satisfying the following conditions:
1. \( T_f \subseteq \mathcal{Y}_p \setminus \{ \alpha \} \) is a set of atomic and ground hash types such that \( \text{pk}(\alpha) \in T_f \) if and only if \( \text{pri}(\alpha) \in T_f \),

2. \( E_f = \{ f(p_1 : \pi_1) = u_1, \ldots, f(p_n : \pi_n) = u_n \} \) is a function specification for \( f \) with respect to \( \Gamma_f : \mathcal{Y}_{pt} \rightarrow \mathcal{Y}_p \) such that
   
   (a) \( \{ \pi_1, \ldots, \pi_n \} \) is a complete set of patterns with respect to composed ground types, i.e., the ground types in the set \( \mathcal{Y}_p \setminus \text{atoms}(\mathcal{Y}_p) \), and
   
   (b) \( \pi_i \) is not deeper than \( \pi_i \) for each \( 1 \leq i \leq n \), i.e., each term position in \( \pi_i \) is also a position in \( \pi_i \).

Moreover, for all \( (f(p : \pi) = u) \in E_f \) one of the following holds:

- \( p = h(q) \) and \( u = h^o(f(q)) \), where \( q \in \mathcal{Y}_{pt} \) and \( a \geq 0 \),
- \( p = \langle q, r \rangle \) and \( u = \{ f(t_1), \ldots, f(t_m) \} \), where \( \text{set}(\text{split}(\langle q, r \rangle)) \subseteq \mathcal{Y}_{pt} \), \( \text{split}(\langle t_1, \ldots, t_m \rangle) = \text{split}(\langle q, r \rangle) \) and \( |t_i| < |\langle q, r \rangle| \) for \( 1 \leq i \leq m \), or
- \( p = \left\langle q \right\rangle_r \) and \( u = \{ \left\langle f(t_1) \right\rangle_{f(r)^a_1}, \ldots, \left\langle f(t_m) \right\rangle_{f(r)^a_m} \} \), \( \text{split}(q) \cup \{ r \} \subseteq \mathcal{Y}_{pt} \), \( a, b \geq 0 \), \( \text{split}(\langle t_1, \ldots, t_m \rangle) = \text{split}(q) \), and \( |t_i| \leq |q| \) for \( 1 \leq i \leq m \).

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

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

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

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

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

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

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

Definition 16 (Protocol transformations). Transforming protocols

Lemma 3.

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

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

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

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

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

Transforming protocols We extend type-based transformations to events, roles and protocols. Transformed events with nil arguments are removed from roles.

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

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

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

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

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

\[
\begin{align*}
\text{K4}(A) &= \text{snd}(A, B, n_A) \cdot \text{rcv}(\lbrack B, T_S, n_A, K_{AB}, X \rbrack_{\text{sh}(A,S)}) \cdot \\
& \text{snd}(X, \lbrack c, T_A \rbrack_{K_{AB}}) \cdot \text{rcv}(\lbrack [t_A] \rbrack_{K_{AB}}) \\
\text{K4}(S) &= \text{rcv}(A, B, N_A) \cdot \text{snd}(\lbrack [B, T_S, N_A, K_{AB}, [A, T_S, k_{AB}]_{\text{sh}(B,S)}]_{\text{sh}(A,S)}) \\
\text{K4}(B) &= \text{rcv}(\lbrack [A, T'_S, K'_{AB}]_{\text{sh}(B,S)}, [c, T_A]_{K'_{AB}}) \cdot \text{snd}(\lbrack [t_A] \rbrack_{K'_{AB}})
\end{align*}
\]
The type-based message transformation $S_{f_4} = (T_{f_4}, E_{f_4})$, where $T_{f_4} = \emptyset$ and $E_{f_4}$ is defined using list concatenation $@$ and $E_0(f)$ from Example 2 as follows.

$$E_{f_4} = \{f_4([X_1, X_2, X_3, X_4, X_5]) \cdot K : \{X_1, X_2, X_3, X_4, X_5\} \cdot \alpha, \alpha\} @ E_0(f_4)$$

Applying $f_4$ to $K_4$ yields $K_5 = f_4(K_4)$ as follows. In this and the next example, we omit roles that are unchanged by the respective transformations.

$$K_5(A) = \text{snd}(A, B, n_A) \cdot \text{rcv}((B, T_S, n_A, K_{AB})_{\text{sh}(A, S)}, X) \cdot \text{snd}(X, \{c, t_A\}_{K_{AB}}) \cdot \text{rcv}((\{t_A\}_{K_{AB}})$$

$$K_5(S) = \text{rcv}(A, B, N_A) \cdot \text{snd}((B, t_S, N_A, K_{AB})_{\text{sh}(A, S)}, \{A, t_S, K_{AB}\}_{\text{sh}(B, S)})$$

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

$$K_3(A) = \text{snd}(A, B, n_A) \cdot \text{rcv}((B, T_S, n_A, K_{AB})_{\text{sh}(A, S)}) \cdot \text{snd}((c, t_A)_{K_{AB}}) \cdot \text{rcv}((\{t_A\}_{K_{AB}})$$

$$K_3(S) = \text{rcv}(A, B, N_A) \cdot \text{snd}((B, t_S, N_A, K_{AB})_{\text{sh}(A, S)}) \cdot \text{snd}((A, t_S, K_{AB})_{\text{sh}(B, S)})$$

$$K_3(B) = \text{rcv}((A, T_S', K_{AB}')_{\text{sh}(B, S)}) \cdot \text{rcv}((c, t_A)_{K_{AB}'})$$

The type-based message transformation $S_{f_3} = (T_{f_3}, E_{f_3})$ is defined by $T_{f_3} = \{\beta_{f_3}, \gamma_{f_3}\}$ and $E_{f_3} = E_0(f_3)$. Applying $f_3$ to K3 yields protocol $K_2 = f_3(K_3)$ where the key confirmation messages have been removed.

$$K_2(A) = \text{snd}(A, B, n_A) \cdot \text{rcv}((B, T_S, n_A, K_{AB})_{\text{sh}(A, S)})$$

$$K_2(B) = \text{rcv}((A, T_S', K_{AB}')_{\text{sh}(B, S)})$$

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

### 3.4 Well-definedness and simulation

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

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

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

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

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

15
Next, we lift deducibility preservation (Theorem 1) to non-ground terms and establish the simulation property. Since protocol descriptions contain non-ground terms, we restrict our attention to simple-keyed protocols, for which the set of (ground) types of the protocol’s terms is simple-keyed. Hereafter, $IK_0$ and $IK'_0$ denote the intruder’s initial knowledge associated with $P$ and $f(P)$, respectively.

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

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

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

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

4 Property language and soundness

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

4.1 Security properties

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

$$\phi ::= A \mid \neg A \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid \forall i. \phi' \mid \exists i. \phi'$$

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

$$A ::= i = j \quad \text{thread } i \text{ and } j \text{ are equal}$$
$$\mid m = m' \quad \text{messages } m \text{ and } m' \text{ are equal}$$
$$\mid \text{role}(i, R) \quad \text{thread } i \text{ executes role } R$$
$$\mid \text{honest}(i, R) \quad \text{the agent playing role } R \text{ in thread } i \text{'s view is honest}$$
$$\mid \text{steps}(i, e) \quad \text{thread } i \text{ has executed event } e$$
$$\mid (i, e) \prec (j, f) \quad \text{thread } i \text{ has executed } e \text{ before thread } j \text{ has executed } f$$
$$\mid \text{secret}(m) \quad \text{the intruder does not know } m$$
To achieve attack preservation, we focus on the fragment of this logic where the predicate \( \text{secret}(m) \) only occurs positively. We call this language \( L_P \). A property is a closed formula of \( L_P \). In examples, we freely use standard abbreviations (e.g., for implication) if there is an equivalent negative normal form in \( L_P \).

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

\[
\begin{align*}
(tr, \text{th}, \sigma, \vartheta) &\models i = j & \text{iff } \vartheta(i) = \vartheta(j) \\
(tr, \text{th}, \sigma, \vartheta) &\models \varphi = \psi & \text{iff } \varphi = \psi \\
(tr, \text{th}, \sigma, \vartheta) &\models \varphi \land \psi & \text{iff } \varphi \land \psi \\
(tr, \text{th}, \sigma, \vartheta) &\models \varphi \lor \psi & \text{iff } \varphi \lor \psi \\
(tr, \text{th}, \sigma, \vartheta) &\models \varphi \rightarrow \psi & \text{iff } \varphi \rightarrow \psi \\
(tr, \text{th}, \sigma, \vartheta) &\models \neg \varphi & \text{iff } \neg \varphi \\
(tr, \text{th}, \sigma, \vartheta) &\models \exists i. \varphi & \text{iff } \exists i. \varphi \\
(tr, \text{th}, \sigma, \vartheta) &\models \forall i. \varphi & \text{iff } \forall i. \varphi \\
\end{align*}
\]

where \( a \prec b \) ("\( a \) 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 \( (tr, \text{th}, \sigma, \vartheta) \not\models \phi \) if \( (tr, \text{th}, \sigma, \vartheta) \models \phi \) does not hold. If \( \phi \) is a closed formula, we write \( (tr, \text{th}, \sigma) \models \phi \) instead of \( (tr, \text{th}, \sigma, \vartheta) \models \phi \).

**Definition 18 (Attack).** We say that a state \( s = (tr, \text{th}, \sigma) \) is an attack on \( \phi \) if \( s \not\models \phi \). The state (attack) \( s \) is well-typed if \( \sigma \) is well-typed.

We extend transformations \( f \) to formulas \( \phi \in L_P \) as follows:

\[
\begin{align*}
&f(i = i') = i = i' & f(\text{secret}(m)) = \text{secret}(f(m)) \\
&f(m = m') = f(m) = f(m') & f(\neg A) = \neg f(A) \\
&f(\text{role}(i, R)) = \text{role}(i, f(R)) & f(\phi_1 \land \phi_2) = f(\phi_1) \land f(\phi_2) \\
&f(\text{honest}(i, R)) = \text{honest}(i, f(R)) & f(\phi_1 \lor \phi_2) = f(\phi_1) \lor f(\phi_2) \\
&f(\text{steps}(i, e)) = \text{steps}(i, f(e)) & f(\exists i. \phi') = \exists i. f(\phi') \\
&f((i, e) \prec (j, e')) = (i, f(e)) \prec (j, f(e')) & f(\forall i. \phi') = \forall i. f(\phi') \\
\end{align*}
\]

**Example 5 (Secrecy and authentication).** Consider the initiator and responder roles of the core Kerberos IV protocol \( K_4 \) as specified in Example 3.

\[
\begin{align*}
K_4(A) &= \text{snd}(A, B, n_A) \cdot \text{rcv}(\{B, T_S, n_A, K_{AB}, X\}_{sh(A, S)}) \cdot \text{snd}(X, \{c, t_A\}_{K_{AB}}) \cdot \text{rcv}(\{t_A\}_{K_{AB}}) \\
K_4(B) &= \text{rcv}(\{A, T_S, K_{AB}'\}_{sh(B, S)}), \{c, T_A\}_{K_{AB}'} \cdot \text{snd}(\{T_A\}_{K_{AB}'}) \\
\end{align*}
\]

We express the secrecy of the session key \( k_{AB} \) for role \( A \) by

\[
\phi_s = \forall i. (\text{role}(i, A) \land \text{honest}(i, [A, B]) \land \text{steps}(i, \text{rcv}(t_2))) \Rightarrow \text{secret}(K_{AB}').
\]

where \( t_2 = \{B, T_S, n_A, K_{AB}, X\}_{sh(A, S)} \) and \( \text{honest}(i, [A, B]) \) abbreviates the obvious conjunction. We abstract this property to verify it on the simplified
protocol \( K_1 = g(K4) \), where \( g = f_2 \circ f_3 \circ f_5 \circ f_4 \) is the composition of all transformations in our running example. Hence, we derive \( \phi_s' = g(\phi_s) \), yielding

\[ \phi_s' = \forall i. (role(i, A) \land honest(i, [A, B]) \land steps(i, rcv(t'_2))) \Rightarrow secret(K'_4) \]

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

\[ \phi_a = \forall i. (role(i, B) \land honest(i, [A, B]) \land steps(i, rcv(u_1, u_2))) \]

\[ \Rightarrow \exists j. role(j, A) \land steps(j, snd(X, \{c, t_A\} K_{AB})) \]

\[ \land (A^j, B^j, K'_{\alpha B}, T'_{\alpha}) = (A^j, B^j, K_{\alpha B}, t'_{\alpha}) \]

where \( u_1 = \{\{A, T'_2, K'_{\alpha B}\}\}_{sh(B,S)} \) and \( u_2 = \{c, T_A\}_{K_{\alpha B}} \). For the simplified protocol \( K3 = f_5 \circ f_4(K4) \), we check the abstracted formula \( \phi_a' = f_5 \circ f_4(\phi_a) \), where \( B \)'s ticket and the associated variable \( X \) of role \( A \) have been removed.

\[ \phi_a' = \forall i. (role(i, B) \land honest(i, [A, B]) \land steps(i, rcv(u_2))) \]

\[ \Rightarrow \exists j. role(j, A) \land steps(j, snd(\{c, t_A\} K_{AB})) \]

\[ \land (A^j, B^j, K'_{\alpha B}, T'_{\alpha}) = (A^j, B^j, K_{\alpha B}, t'_{\alpha}) \]

### 4.2 Soundness

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

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

Our soundness result is therefore restricted to a subset of \((P, f)\)-safe formulas of \( L_P \). We first define some auxiliary notions. Let \( T_{eq}^+(\phi) \) be the set of pairs \((m, m')\) such that the equation \( m = m' \) occurs positively in \( \phi \) and let \( T_{ext}(\phi) \) be the set of events \( s(m), s'(m') \) such that \((i, s(m)) \prec (j, s'(m')) \) or \( steps(i, s(m)) \) occurs (positively) in \( \phi \).

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

1. \( m \sigma \neq m' \sigma \) implies \( f(m\sigma) \neq f(m'\sigma) \) for all \((m, m') \in T_{eq}^+(\phi) \) and well-typed ground substitutions \( \sigma \),
2. \( m \neq m' \) implies \( f(m) \neq f(m') \) for all \( s(m) \in T_{ext}^+(\phi) \) and \( s(m') \in Event^4_P \), and
3. \( f(m) \neq \text{nil} \) for all \( s(m) \in T_{ext}(\phi) \).
Table 1. Experimental verification results for Kerberos (times in seconds); the abstraction level increases from left to right columns; (*) denotes highest abstraction level for marked properties.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>K4/6</th>
<th>K5/6</th>
<th>K3/6</th>
<th>K2/6</th>
<th>K1/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td>sec</td>
<td>aut</td>
<td>kc</td>
<td>sec</td>
<td>aut</td>
</tr>
<tr>
<td>Time [sec]</td>
<td>1.45</td>
<td>1.31</td>
<td>1.16</td>
<td>20.65</td>
<td>18.27</td>
</tr>
<tr>
<td>#Clauses/1000</td>
<td>15.4</td>
<td>13.8</td>
<td>12.1</td>
<td>188.9</td>
<td>165.6</td>
</tr>
<tr>
<td>#Atoms/1000</td>
<td>2.0</td>
<td>1.9</td>
<td>1.9</td>
<td>33.0</td>
<td>32.9</td>
</tr>
</tbody>
</table>

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

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

4.3 Experimental results

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

The columns denote the protocols and the properties verified. We grouped the properties into three classes: session key secrecy from the perspective of each role (sec), authentication properties involving a Kerberos server (aut), and key confirmation (kc). Those columns where the highest degree of abstraction for a given property class is achieved are marked with a star (*). The rows show the verification time and the number of clauses and atoms of the SAT encoding (in thousands). The verification time is dominated by the encoding into a SAT problem whereas the SAT solving time is negligible.
We observe a slowdown from K4/6 to K5/6. We attribute this to the unencrypted responder ticket in K5/6, which increases the intruder’s possibilities to interfere with the ticket variable X. The performance on K3/6 is similar to the one on K4/6. More interesting are the performance gains obtained by the further abstractions and the overall speedups that we achieve for the protocols K4 and K5. For example, verifying secrecy on K1/6 is 148 times faster than on K5/6 and still 10.4 times faster than on K4/6.

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

5 Related work

We can classify existing work on protocol transformations into syntactic and semantic approaches. Syntactic approaches use syntactic criteria to delimit a class of transformations for which soundness can be established a priori. Hui and Lowe [11] define several kinds of transformations similar to ours with the aim improving the performance of the CASPER/FDR model checker. They prove soundness of each kind of transformations based on general soundness criteria for secrecy and authentication. Their protocol model is restricted to atomic keys and they establish their results only for ground messages. We work in a more general setting and discuss in detail the non-trivial issue of handling terms with variables as they appear in protocol specifications. Other works [15,7,6] propose a set of syntactic transformations without however formally establishing their soundness.

Semantic approaches generally cover a larger class of transformations, but each transformation requires a separate proof for its justification. Examples are classical refinement and using abstract channels with security properties [16,4] and Guttman’s protocol transformations based on strand spaces [9,8]. Sprenger and Basin [16] have recently proposed a refinement strategy for security protocols that spans several different abstraction levels (including, e.g., confidential and authentic channels). The transformations in the present paper belong to their most concrete level of cryptographic protocols. Guttman [9,8] studies the preservation of security properties by a rich class of protocol transformations in the strand space model. His approach to property preservation is based on the simulation of protocol analysis steps instead of execution steps. Each analysis step explains the origin of a received message. However, he does not provide syntactic conditions for the transformations’ soundness.
6 Conclusions

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

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

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

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

References


A Proofs for deducibility preservation (Section 3.2)

A.1 Proof of Lemma 1

We first introduce the notion of plaintext subterms.

**Definition 20 (Plaintext).** We define the set $\text{plain}(t)$ of plaintext subterms of a term $t$ by

\[
\begin{align*}
\text{plain}(u) & = \{u\} \quad \text{if } u \text{ is a variable or an atom} \\
\text{plain}(h(u)) & = \text{plain}(u) \\
\text{plain}([u_1, u_2]) & = \text{plain}(u_1) \cup \text{plain}(u_2) \\
\text{plain}([u]^k) & = \text{plain}(u)
\end{align*}
\]

From Definition 20 and Definition 2, the following lemma is trivial.

**Lemma 6.** Let $u, t \in T$ be ground terms. If $t$ is simple and $\text{plain}(u) \subseteq \text{plain}(t)$ then $u$ is simple.

Next we show that a composed simple term cannot be mapped to an asymmetric key by message transformations.

**Lemma 7.** Let $f : T \to T$ be a message transformation and let $t \in T$ be a composed simple term. Then $f(t)$ is not an asymmetric key.

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

- If $t$ is an atom or a variable then the lemma trivially holds.
- If $t = h(u)$ then either $nf(t) = \text{nil}$ or $nf(t) = h(nf(u))$.
  - If $nf(t) = \text{nil}$ then $f(t) = \text{nil}$ which is not an asymmetric key,
  - otherwise, we have $nf(t) = h(nf(u))$. Therefore, it immediately follows from Definition 6 that $f(t)$ is not an asymmetric key.
- If $t = [u_1, u_2]$ then $f(t) = nf((f(t_1), \ldots, f(t_n)))$, where

\[
\forall i \in \{1, \ldots, n\} \quad |t_i| < |t|,
\]

\[
\text{split}([t_1, \ldots, t_n]) \subseteq \text{split}(t)
\]

We consider two cases:

- If $f(t) = \text{nil}$ then $f(t)$ is not an asymmetric key.
- If $f(t) \neq \text{nil}$ then there exists $i \in \{1, \ldots, n\}$ such that $f(t_i) \neq \text{nil}$.
  * If $t_i$ is not composed then since $t$ is a simple, $t_i$ is not an asymmetric key. Moreover, since $f(t_i) \neq \text{nil}$ we have $f(t_i) = t_i$. Hence $f(t_i)$ is not an asymmetric key and thus $f(t)$ is not an asymmetric key.
  * If $t_i$ is composed then we have $\text{plain}(t_i) \subseteq \text{plain}(t)$. Since $t$ is a simple, by Lemma 6 we have $t_i$ is also a simple. Thus by induction hypothesis we have $f(t_i)$ is not an asymmetric key for all $i \in \{1, \ldots, n\}$. Therefore, $f(t)$ is not an asymmetric key.
- If $t = \{u\}_k$ then $f(t) = \text{nf}(\{\langle f(t_1)\rangle_{f(k)^a}, \ldots, \langle f(t_n)\rangle_{f(k)^a}\}_k)$, where

\[
\forall i \in \{1, \ldots, n\}, |t_i| \leq |u|,
\text{split}(\langle t_1, \ldots, t_n \rangle) \subseteq \text{split}(u)
\]

By a similar reasoning as for pairs, we have that $f(t)$ is not an asymmetric key.

This completes the proof of the lemma. \qed

We now can prove Lemma 1.

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

**Proof.** We consider two cases:

- If $t$ is not composed, then there are three cases:
  - If $t = \text{pri}(a)$ then by Definition 6, $f(t) = f(t^{-1}) = \text{nil}$ or $f(t) = t$.
    - If $f(t) = f(t^{-1}) = \text{nil}$ then $f(t^{-1}) = \text{nil} = f(t^{-1})$.
    - If $f(t) = t$ then $f(t)^{-1} = f(\text{pk}(a)) = \text{pk}(a) = \text{pri}(a)^{-1} = f(t)^{-1}$.
  - If $t = \text{pk}(a)$ then similarly as above, we have $f(t^{-1}) = f(t)^{-1}$.
  - If $t$ is not an asymmetric key, then $t^{-1} = t$. Hence by Definition 6 we have two cases:
    - If $f(t) = t$ then $f(t)^{-1} = f(t) = t = f(t)^{-1}$.
    - If $f(t) = \text{nil}$ then $f(t)^{-1} = f(t) = \text{nil} = f(t)^{-1}$.

- If $t$ is a composed term then since $t$ is a simple, by Lemma 7 we have $f(t)$ is not an asymmetric key. Therefore, $f(t)^{-1} = f(t)$. Moreover, we have $t^{-1} = t$. Hence $f(t^{-1}) = f(t) = f(t)^{-1}$.

This completes the proof of the lemma. \qed

**A.2 Lemmas related to normal forms and pair splitting**

**Lemma 8.** Let $T$ be a set of terms such that $\text{nil} \in T$ and $t \in T$. If one of the following holds

- $T \vdash t$ and $t$ is a variable or an atom,
- $t = \langle t_1, \ldots, t_n \rangle$, where for all $i \in \{1, \ldots, n\}$
  - $t_i$ is in normal form, and
  - $T \vdash t_i$,
- $t = \{\langle t_1\rangle_{k^1}, \ldots, \langle t_n\rangle_{k^n}\}_k$, where for all $i \in \{1, \ldots, n\}$
  - $t_i$ and $k$ are in normal form,
  - $T \vdash t_i$ and $T \vdash k$, and
  - $a_i \geq 0$ and $b \geq 0$,

then $T \vdash \text{nf}(t)$. 

24
Proof. We prove the proposition by case distinction on \( t \).

- If \( t \) is a variable or an atom, then \( \text{nf}(t) = t \) or \( \text{nf}(t) = \text{nil} \). Since we have \( \{\text{nil}, t\} \subseteq T \), it is obvious that \( T \vdash \text{nf}(t) \).

- If \( t = \langle t_1, \ldots, t_n \rangle \) and for all \( i \in \{1, \ldots, n\} \)
  - \( t_i \) is in normal form, and
  - \( T \vdash t_i \)

then by Definition 5 we have

\[
\text{nf}(t) = \langle t_{i_1}, \ldots, t_{i_m} \rangle
\]

where \( 1 \leq i_1 < \ldots < i_m \leq n \) and \( t_j \neq \text{nil} \leftrightarrow j \in \{i_1, \ldots, i_m\} \) for all \( j \in \{1, \ldots, n\} \). Since we have

1. \( \text{nil} \in T \),
2. \( \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\} \), and
3. \( T \vdash t_i \) for all \( i \in \{1, \ldots, n\} \),

it is obvious that \( T \vdash \text{nf}(t) \).

- If \( t = \{\langle t_1 \rangle_{k^{a_1}}, \ldots, \langle t_n \rangle_{k^{a_n}} \} \}_{k^b} \), and for all \( i \in \{1, \ldots, n\} \)
  - \( t_i \) and \( k \) are in normal form
  - \( T \vdash t_i \) and \( T \vdash k \), and
  - \( a_i \geq 0 \) and \( b \geq 0 \),

then since \( k \) is in normal form, we consider two cases:

  - If \( k = \text{nil} \) then \( \text{nf}(t) = \text{nf}(\langle t_1, \ldots, t_n \rangle) \). As above, we have \( T \vdash \text{nf}(t) \).
  - If \( k \neq \text{nil} \) then we have

\[
\text{nf}(t) = \{\langle t_{i_1} \rangle_{k^{a_1}}, \ldots, \langle t_{i_m} \rangle_{k^{a_m}} \}_{k^b}
\]

where \( 1 \leq i_1 < \ldots < i_m \leq n \) and \( t_j \neq \text{nil} \leftrightarrow j \in \{i_1, \ldots, i_m\} \) for all \( j \in \{1, \ldots, n\} \). Moreover, since we have

1. \( T \vdash t_i \) for all \( i \in \{1, \ldots, n\} \),
2. \( \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\} \), and
3. \( T \vdash k \),

it is easy to see that \( T \vdash \text{nf}(t) \).

This completes the proof of the lemma. \( \square \)

In order to prove deducibility preservation for our class of protocol transformations, we first show that if the intruder learns a transformed term, he can then learn all the transformed components of the term.

**Lemma 9.** Let \( f \) be a message transformation on \( T \) and \( u \in T \). Then for all \( t \in \text{set}(\text{split}(u)) \), we have \( \{f(u), \text{nil}\} \vdash f(t) \)

Proof. We first prove the lemma for all terms \( u \in \text{nf}(T) \) by induction on \( |u| \).

- If \( |u| = 1 \), then \( \text{set}(\text{split}(u)) = \{u\} \). Hence \( t \in \text{set}(\text{split}(u)) \) implies that \( t = u \). Thus \( \{f(u), \text{nil}\} \vdash f(t) \).
Lemma 11. For all terms $t$, if $t \in \text{set}(\text{split}(u))$, then there exists $t' \in \text{set}(\text{split}(u))$ such that $t \in \text{set}(\text{split}(u)) \cup \{\text{nil}\}$.

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 $t \in \text{set}(\text{split}(u))$ implies that $t \in \text{set}(\text{split}(u_i))$ for some $i \in \{1, 2\}$. By induction hypothesis, we have

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

This together with Definition 5 imply that

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

This completes the proof of the lemma.

Lemma 10. For all terms $t, u \in \mathcal{T}$, if $t \in \text{set}(\text{split}(u))$ then we have

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

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

- If $|u| > 1$ and $u$ is not a pair, then $\text{set}(\text{split}(u)) = \{u\}$. Hence as above, the lemma holds.
- If $|u| > 1$ and $u = \langle u_1, u_2 \rangle$, then by Definition 6, we have

$$f(u) = \text{nf}(\langle f(t_1), \ldots, f(t_n) \rangle)$$

for all $j \in \{1, \ldots, n\}$, $\text{split}(t_j) \subseteq \text{split}(u)$

$$\text{set}(\text{split}(\langle t_1, \ldots, t_n \rangle)) = \text{set}(\text{split}(u))$$

Moreover, by Definition 6, we have

$$\text{nf}(\langle f(t_1), \ldots, f(t_n) \rangle) \subseteq \text{split}(u)$$

Since $t \in \text{set}(\text{split}(u))$, by (4) we have $t \in \text{set}(\text{split}(\langle t_1, \ldots, t_n \rangle))$. Hence there exists $j \in \{1, \ldots, n\}$ such that $t \in \text{set}(\text{split}(t_j))$. Since $u \in \text{nf}(\mathcal{T})$ and $u$ is a pair, $u$ does not contain nil. This together with $\text{split}(t_j) \subseteq \text{split}(u)$ imply that $t_j \in \text{nf}(\mathcal{T})$. Moreover, we have $|t_j| < |u|$. Hence by induction hypothesis, we have $\{f(t_j), \text{nil}\} \vdash f(t)$. This together with (4) imply that $\{f(u), \text{nil}\} \vdash f(t)$.

If $u \notin \text{nf}(\mathcal{T})$, then since $t \in \text{set}(\text{split}(u))$, we have $\text{nf}(t) \in \text{split}(\text{nf}(u))$ or $\text{nf}(t) = \text{nil}$. In both cases, we have

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

Moreover, by Definition 6, we have

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

(6)

By (5) and (6) we have $\{f(u), \text{nil}\} \vdash f(t)$.

Lemma 11. For all terms $t, u \in \mathcal{T}$, if $t \in \text{set}(\text{split}(u))$, then there exists $t' \in \text{set}(\text{split}(u))$ such that $t \in \text{set}(\text{split}(\text{nf}(t'))) \cup \{\text{nil}\}$.

Proof. We prove the lemma by induction on $u$.
– If \( u \) is neither a pair nor an encryption, then \( nf(u) \) is not a pair. Hence
\[
set(split(u)) = \{ u \} \quad \text{and} \quad set(split(nf(u))) = \{ nf(u) \}.
\]
Therefore, we have \( t \in set(split(nf(u))) \) implies that \( t = nf(u) \). We choose \( t' = u \) which satisfies
\[
t \in set(split(nf(t'))) \cup \{ nil \}.
\]
– If \( u = [u']_k \), then by Definition 5, there are two cases:
  • If \( nf(u) \) is an encryption, then it is similar as above, the lemma holds for this case.
  • If \( nf(u) = nf(u') \), then \( t \in set(split(nf(u))) \) yields \( t \in set(split(nf(u'))) \).

Moreover, by induction hypothesis, there exists \( w \in set(split(u')) \) such that \( t \in set(split(nf(w))) \cup \{ nil \} \). Since \( w \in set(split(u')) \), by Lemma 10, we have
\[
set(split(nf(w))) \cup \{ nil \} \subseteq set(split(nf(u'))) \cup \{ nil \}
\]
Hence we have
\[
t \in set(split(nf(u'))) \cup \{ nil \}
\]
We now choose \( t' = u \). Thus we have \( t' \in set(split(u)) = \{ u \} \). Since \( nf(u') = nf(u) \), by (7) we obtain \( t \in set(split(t')) \cup \{ nil \} \).

– If \( u = \langle u_1, u_2 \rangle \) then by Definition 5, we have
\[
set(split(nf(u))) = set(split(nf(u_{i_1}))) \cup set(split(nf(u_{i_2}))), \quad \text{where} \quad i_1, i_2 \in \{ 1, 2 \}
\]
Hence \( t \in set(split(nf(u))) \) implies that \( t \in set(split(nf(u_{i_j}))) \) for \( i \in \{ i_1, i_2 \} \).
By induction hypothesis, there exists \( t' \in set(split(u_i)) \) such that
\[
t \in set(split(nf(t'))) \cup \{ nil \}
\]
Moreover, since \( set(split(u)) = set(split(u_{i_1})) \cup set(split(u_{i_2})) \), by (8) and (9) we also have \( t' \in set(split(u)) \).

This completes the proof of the lemma. \( \square \)

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

**Lemma 12.** Let \( f \) be a protocol transformation on \( T \), \( u \in T \), \( T \subseteq T \) and \( nil \in T \). Then
\[
(\forall t \in set(split(u)). T \vdash f(t) \Rightarrow T \vdash f(u))
\]

*Proof*. We prove the lemma for all \( u \in nf(T) \) by induction on the size of \( u \).

Assume that
\[
(\forall t \in set(split(u)). T \vdash f(t)) \tag{10}
\]
We need to show that
\[
T \vdash f(u) \tag{11}
\]

– If \( |u| = 1 \), then \( set(split(u)) = \{ u \} \). By (10), we have \( T \vdash f(u) \).
If $|u| > 1$ and $u$ is not a pair, then $set(split(u)) = \{u\}$. Hence as above, the lemma holds.

If $|u| > 1$, then $u = \langle u_1, u_2 \rangle$. By Definition 6, we have

$$f(u) = nf(\langle f(t_1), \ldots, f(t_n) \rangle)$$  \hspace{1cm} (12)

$$split(\langle t_1, \ldots, t_n \rangle) \subseteq split(u)$$  \hspace{1cm} (13)

From (10) and (13) it follows that $T \vdash f(t)$ for all $t \in set(split(t_i))$ and $i \in \{1, \ldots, n\}$. Since $u \in nf(T)$, we have $t_i \in nf(T)$ for all $i \in \{1, \ldots, n\}$. Hence, by induction hypothesis, we have $T \vdash f(t_i)$. Moreover, since $f(t_i)$ is in normal form for all $i \in \{1, \ldots, n\}$, by Lemma 8 and (12) we have $T \vdash f(u)$.

If $u \not\in nf(T)$, then by Lemma 11, for all $t \in set(split(nf(u)))$, there exists $t' \in set(split(u))$ such that

$$t \in set(split(nf(t'))) \cup \{\text{nil}\}$$  \hspace{1cm} (14)

By assumption 10, we have $T \vdash f(t')$. Therefore, we have $T \vdash f(nf(t'))$. This together with (14) and Lemma 9 imply that $T \vdash f(t)$. This, since $nf(u) \in nf(T)$, implies that $T \vdash f(nf(u))$. Moreover, we have $f(u) = f(nf(u))$. Hence $T \vdash f(u)$. \hfill \Box

The following lemma is a consequence of those two above lemmas.

**Lemma 13.** Let $f$ be a protocol transformation on $T$ and $\{u, t\} \subseteq T$. Then $set(split(t)) \subseteq set(split(u))$ implies that $\{f(u), \text{nil}\} \vdash f(t)$.

**Proof.** By Lemma 9, for all $p \in set(split(u))$ we have

$$\{f(u), \text{nil}\} \vdash f(p)$$  \hspace{1cm} (15)

Moreover, since $set(split(t)) \subseteq set(split(u))$, by (15), for all $q \in set(split(t))$, we have

$$\{f(u), \text{nil}\} \vdash f(q)$$  \hspace{1cm} (16)

By (16) and Lemma 12, we have $\{f(u), \text{nil}\} \vdash f(t)$. \hfill \Box

### A.3 Proof of deducibility preservation

Before we prove the deducibility preservation theorem, we show that in minimal-size derivations of $T \vdash u$ contain only subterms of terms in $T \cup \{u\}$.

**Definition 21 (Minimal derivation trees).** A derivation tree of $T \vdash u$ is minimal iff its number of nodes is minimal.

**Definition 22 (Local derivation trees).** A derivation tree of $T \vdash u$ is local iff it only involves terms in $St(T \cup \{u\})$.
Lemma 14 (Locality). If $D$ is a minimal derivation tree of $T \vdash u$ then $D$ is local. Moreover, if the last rule applied is a decomposition or an axiom then $u \in St(T)$.

Proof. We proceed by induction on $D$ depending on the last rule that has been applied.

- If the last rule is an axiom then $u \in T$. Hence the tree is local.
- If the last rule is a composition then we consider two cases:
  - If $T \vdash t$ and $u = h(t)$ then the derivation tree $D'$ of $T \vdash t$ is minimal. Hence by induction hypothesis, $D'$ is local. Since $St(t) \subset St(u)$, $D$ is also local.
  - If $T \vdash u_1, T \vdash u_2$ and $u = f(u_1, u_2)$ for $f \in \{<, \{\} \}$ then the derivation trees $D_1$ of $T \vdash u_1$ and $D_2$ of $T \vdash u_2$ are minimal. Hence by induction hypothesis, $D_1$ and $D_2$ are local. Since $St(u_1) \cup St(u_2) \subset St(u)$, $D$ is also local.
- If the last rule is a decomposition then it is enough to consider two cases:
  - If the rule is a decryption, then let $D'$ be the derivation tree of $T \vdash \{\} | k$. Since $D$ is minimal, so is $D'$. Since $D'$ is minimal, $D'$ ends with a decomposition rule. Hence by induction hypothesis, $\{\} | k \in St(T)$. In particular, $u \in St(T)$.
  - If the rule is $Proj_1$ then the lemma holds for this case by a similar reasoning as for the case of symmetric decryption.

This completes the proof of the lemma. □

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

Proof. Without loss of generality, we consider a minimal derivation tree $D$ of $T \vdash u$ and proceed by induction on $D$ depending on the last rule that has been applied.

- If the last rule is an axiom, then $u \in T$. Hence $f(u) \in f(T)$ which implies that $f(T) \cup \{\text{nil}\} \vdash f(u)$.
- If the last rule is a composition, then there are three cases:
  - $T \vdash u'$ and $u = h(u')$. By induction hypothesis, we have
    \begin{equation}
    f(T) \cup \{\text{nil}\} \vdash f(u')
    \end{equation}
    Moreover, we have $nf(u) = \text{nil}$ or $nf(u) = h(nf(u'))$. If $nf(u) = \text{nil}$, then by Definition 6, we have $f(u) = f(nf(u)) = \text{nil}$. Hence $f(T) \cup \{\text{nil}\} \vdash f(u)$. We consider the case that $nf(u) = h(nf(u'))$. By Definition 6, we have
    \begin{equation}
    f(nf(u)) \in \{nf(h^a(f(nf(u')))) \mid a \geq 0\} \cup \{\text{nil}\}
    \end{equation}

29
If \( f(nf(u)) = \text{nil} \), then it is obvious that \( f(T) \cup \{ \text{nil} \} \vdash f(u) \). Otherwise, since \( f(u') = f(nf(u')) \), by (17) we have \( f(T) \cup \{ \text{nil} \} \vdash f(nf(u')) \). Hence \( f(T) \cup \{ \text{nil} \} \vdash h^\alpha(f(nf(u'))) \). Since \( f(nf(u')) \) is in normal form, either

\[
\begin{align*}
nf(h^\alpha(f(nf(u')))) &= \text{nil}, \quad \text{or} \\
nf(h^\alpha(f(nf(u')))) &= h^\alpha(f(nf(u')))
\end{align*}
\]

In both cases, we have \( f(T) \cup \{ \text{nil} \} \vdash nf(h^\alpha(f(nf(u')))) \). Together with \( f(u) = f(nf(u)) \) and (18), we obtain \( f(T) \cup \{ \text{nil} \} \vdash f(u) \).

\( T \vdash u_1, T \vdash u_2 \) and \( u = \langle u_1, u_2 \rangle \). By induction hypothesis, we have

\[
\begin{align*}
f(T) \cup \{ \text{nil} \} &\vdash f(u_1) \\
f(T) \cup \{ \text{nil} \} &\vdash f(u_2)
\end{align*}
\]

If \( nf(u_1) = \text{nil} \) then \( nf(u) = nf(u_2) \). By Definition 6, we have \( f(u_2) = f(nf(u_2)) \) and \( f(u) = f(nf(u)) \). Hence by (19), we obtain

\[
f(T) \cup \{ \text{nil} \} \vdash f(u)
\]

It is similar for the case \( nf(u_2) = \text{nil} \). Now we assume that \( nf(u_i) \neq \text{nil} \) for \( i \in \{1, 2\} \). Hence \( nf(u) = \langle nf(u_1), nf(u_2) \rangle \). By Definition 6, we have

\[
f(nf(u)) = nf((f(t_1), \ldots, f(t_n)))
\]

where

\[
split(t_1, \ldots, t_n) \sqsubseteq split((nf(u_1), nf(u_2)))
\]

By (21), for all \( i \in \{1, \ldots, n\} \) and \( p \in \text{set}(split(t_i)) \), there exists \( j \in \{1, 2\} \) such that

\[
p \in \text{set}(split(nf(u_j)))
\]

By (22) and Lemma 9, we have

\[
\{ f(nf(u_j), \text{nil} \} \vdash f(p)
\]

By (19), we have \( f(T) \cup \{ \text{nil} \} \vdash f(u_j) \). Since \( f(u_j) = f(nf(u_j)) \), we have \( f(T) \cup \{ \text{nil} \} \vdash f(nf(u_j)) \). This together with (23) imply that

\[
f(T) \cup \{ \text{nil} \} \vdash f(p)
\]

By (24) and Lemma 12, for all \( i \in \{1, \ldots, n\} \), we have \( f(T) \cup \{ \text{nil} \} \vdash f(t_i) \). This together with (20) and Lemma 8 yield that \( f(T) \cup \{ \text{nil} \} \vdash f(nf(u)) \). Since \( f(u) = f(nf(u)) \), we obtain \( f(T) \cup \{ \text{nil} \} \vdash f(u) \).

\( T \vdash u', T \vdash k \) and \( u = \|u'\|_k \). By induction hypothesis, we have

\[
\begin{align*}
f(T) \cup \{ \text{nil} \} &\vdash f(u') \\
f(T) \cup \{ \text{nil} \} &\vdash f(k)
\end{align*}
\]

By Definition 6, we have

\[
\begin{align*}
f(u') &= f(nf(u')) \\
f(k) &= f(nf(k))
\end{align*}
\]

30
Moreover, by Definition 5, one of the following holds:

\[ nf(u) = nf(u'), \]
\[ nf(u) = \{ nf(u') \}_n(k) \]

Hence by (25) and (26), the theorem holds for the first case. It remains to consider the case that \( nf(u) = \{ nf(u') \}_n(k) \). By Definition 6, we have

\[ f(nf(u)) = nf(\{ f(t_1) \}_{nf(k)}^{nf(k)}, \ldots, \{ f(t_n) \}_{nf(k)}^{nf(k)}) \]
\[ split(t_1, \ldots, t_n) \subseteq split(nf(u')) \]

We now show that \( f(T) \cup \{ nil \} \vdash f(nf(u)) \). Given (25) and Lemma 8, it is enough to show that for all \( j \in \{ 1, \ldots, n \} \),

\[ f(T) \cup \{ nil \} \vdash f(t_j) \]  \hspace{1cm} (27)

Let \( j \in \{ 1, \ldots, n \} \) be arbitrary. Since \( split(t_j) \subseteq split(nf(u')) \), we have \( set(split(t_j)) \subseteq set(split(nf(u'))) \). Hence by Lemma 13, it follows that \( \{ f(nf(u')), nil \} \vdash f(t_j) \). Together with (25) and (26), this implies that (27) holds. Hence we have \( f(T) \cup \{ nil \} \vdash f(nf(u)) \). Since \( f(u) = f(nf(u)) \), we also have \( f(T) \cup \{ nil \} \vdash f(u) \).

- If the last rule is a decomposition, then without loss of generality, we only consider two cases:
  - \( T \vdash \langle u, u' \rangle \). By induction hypothesis, we have
    \[ f(T) \cup \{ nil \} \vdash f(\langle u, u' \rangle) \]  \hspace{1cm} (28)
    Moreover, by Lemma 13, we have \( \{ f(\langle u, u' \rangle), nil \} \vdash f(u) \). This together with (28) imply that \( f(T) \cup \{ nil \} \vdash f(u) \).
  - \( T \vdash \{ u \}_k \) and \( T \vdash k^{-1} \). By induction hypothesis we have
    \[ f(T) \cup \{ nil \} \vdash f(\{ u \}_k) \]
    \[ f(T) \cup \{ nil \} \vdash f(k^{-1}) \]  \hspace{1cm} (29)
    Moreover, by Definition 5 and Definition 6, we have \( f(T) \cup \{ nil \} \vdash f(u) \) for the case that \( nf(\{ u \}_k) = nf(u) \). We consider the case that \( nf(\{ u \}_k) = \{ nf(u) \}_n(k) \). Since \( D \) is minimal, \( T \vdash \{ u \}_k \) is the conclusion of a decomposition or an axiom. Hence by Lemma 14, we have \( \{ u \}_k \in St(T) \). Since \( T \) is a simple-keyed, \( k \) is a simple. Hence by Lemma 1, we have \( f(k^{-1}) = f(k)^{-1} \). Since \( f(k) = f(nf(k)) \), by (29) we have
    \[ f(T) \cup \{ nil \} \vdash f(\{ nf(u) \}_n(k)) \]
    \[ f(T) \cup \{ nil \} \vdash f(nf(k))^{-1} \]  \hspace{1cm} (30)
    We have to show that \( f(T) \cup \{ nil \} \vdash f(nf(u)) \). By Definition 6, we have
    \[ f(\{ u' \}_k) = nf(\{ f(t_1) \}_{f(k')}^{f(k')}, \ldots, f(t_n) \}_{f(k')}^{f(k')}) \]  \hspace{1cm} (31)
\[ split(\langle t_1, \ldots, t_n \rangle) \subseteq split(u'), \quad \text{and} \]
\[ set(split(\langle t_1, \ldots, t_n \rangle)) = set(split(u')), \]  
(32)

where \( u' = nf(u) \) and \( k' = nf(k) \).

We now show that for all \( i \in \{1, \ldots, n\} \), we have

\[ f(T) \cup \{nil\} \vdash f(t_i) \]  
(33)

Let \( i \in \{1, \ldots, n\} \). If \( f(t_i) = \text{nil} \) then we have \( f(T) \cup \{\text{nil}\} \vdash f(t_i) \). Assume that \( f(t_i) \neq \text{nil} \). Then since \( f(t_i) \) is in normal form for all \( i \in \{1, \ldots, n\} \), by (31) we have two cases:

* If \( f(k') = \text{nil} \) then

\[ f(\{u'\}_k') = \langle t_{i_1}, \ldots, t_{i_m} \rangle, \]

* If \( f(k') \neq \text{nil} \) then

\[ f(\{u'\}_k') = nf(\{f(t_{i_1})\}_k')^{a_{i_1}}, \ldots, \{f(t_{i_m})\}_k')^{a_{i_m}} \]

where

\[ 1 \leq i_1 < \ldots < i_m \leq n, \quad \text{and} \]
\[ \forall j \in \{1, \ldots, n\}, f(t_j) \neq \text{nil} \iff j \in \{i_1, \ldots, i_m\} \]  
(34)

Since \( f(t_i) \neq \text{nil} \), by (34) we have \( i \in \{i_1, \ldots, i_m\} \). In both cases, we obtain \( f(T) \cup \{\text{nil}\} \vdash f(t_i) \) from (30).

Since \( f(u) = f(nf(u)) \), we need to show that \( f(T) \cup \{\text{nil}\} \vdash f(nf(u)) \).

By Lemma 12, it is sufficient to show that for all \( v \in set(split(nf(u))) \),

\[ f(T) \cup \{\text{nil}\} \vdash f(v) \]  
(35)

Let \( v \in set(split(nf(u))) \). By (32) we have \( v \in set(split(\langle t_1, \ldots, t_n \rangle)) \).

Hence there exists \( j \in \{1, \ldots, n\} \) such that \( v \in set(split(t_j)) \). This by Lemma 9 yields \( \{f(t_j), \text{nil}\} \vdash f(v) \). Together with (33) we obtain (35).

This completes the proof of the theorem.  
\[ \square \]
B Proofs for type-based transformations (Section 3.3)

B.1 Proof of Lemma 3

We first prove that every composed term \( t \) of a type \( \tau \) and pattern of the form \( p : \pi \), whenever \( \pi \) matches \( \tau \) we also have \( p \) matches \( t \).

Lemma 15. Let \( P \) be a splitting protocol, \( \Gamma : V_{pt} \rightarrow \mathcal{V}_P \) a typing environment, \( p \in \mathcal{P} \) be a linear pattern such that \( \Gamma \vdash p : \pi \) and one of the following holds:

- \( p \in V_{pt} \),
- \( p = h(q) \) and \( q \in V_{pt} \),
- \( p = \langle q, r \rangle \) and \( \text{set(split((q, r))))} \subseteq V_{pt} \), or
- \( p = \{q\}_r \) and \( \text{set(split(q))} \cup \{r\} \subseteq V_{pt} \).

Then, for all composed terms \( t \in \mathcal{M}^2_P \), \( \Gamma_P \vdash t : \tau \) and \( \theta : \mathcal{V}_y \rightarrow \mathcal{V}_P \), \( \pi \theta = \tau \) implies that there exists a substitution \( \sigma : \text{vars}(p) \rightarrow \mathcal{M}^2_P \) such that \( p \sigma = t \).

Proof. We prove the lemma by induction on the structure of \( p \).

- If \( p \) is a pattern-variable, then we define \( \sigma = [p/t] \), hence \( p \sigma = t \).
- If \( p = h(q) \), then \( q \) must be a pattern-variable. Since \( \Gamma \vdash p : \pi \), we have
  \[
  \pi = h(\pi') \\
  \Gamma \vdash q : \pi'
  \]
  (36)
  Since \( \pi \theta = \tau \), by (36), we have \( h(\pi' \theta) = \tau \). Hence \( \tau = h(\tau') \), where \( \tau' = \pi' \theta \).

Moreover, since \( \Gamma_P \vdash t : \tau \) and \( t \) is composed, we have

\[
\Gamma_P \vdash t : \tau'
\]

We define \( \sigma = [q/t'] \). This implies that \( q \sigma = t' \) and hence \( p \sigma = t \).

- If \( p = \langle p_1, p_2 \rangle \), then since \( \Gamma \vdash p : \pi \), we have
  \[
  \pi = \langle \pi_1, \pi_2 \rangle \\
  \Gamma \vdash p_i : \pi_i \text{ for } i \in \{1, 2\}
  \]
  (37)
  Since \( \pi \theta = \tau \), by (37) we have
  \[
  \tau = \langle \tau_1, \tau_2 \rangle \\
  \tau_i = \pi_i \theta \text{ for } i \in \{1, 2\}
  \]
  (38)

Since \( \Gamma_P \vdash t : \tau \) and \( t \) is composed, by (38) we have

\[
\Gamma_P \vdash t_i : \tau_i \text{ for } i \in \{1, 2\}
\]
  (39)

For each \( i \in \{1, 2\} \), we consider two cases:

- If \( p_i \) is a pattern-variable, then we define \( \sigma_i = [p_i/t_i] \).
- If $p_i$ is not a pattern-variable, then by assumption, $p_i$ is a pair and $\text{set}(\text{split}(p_i)) \subseteq V_{\rho}$. Hence by (37), we have $\pi_i$ is a pair. This together with (38) imply that $\tau_i$ is a pair. By (39) and the fact that $P$ is splitting, it follows that $t_i$ cannot be a variable. Hence $t_i$ must be a pair. Thus by induction hypothesis, there exists $\sigma_i : \text{vars}(p_i) \rightarrow \mathcal{M}_P^\cap$ such that $p_i\sigma_i = t_i$.

Since $p$ is linear, $\text{vars}(p_1) \cap \text{vars}(p_2) = \emptyset$. Hence, we define $\sigma : \text{vars}(p) \rightarrow \mathcal{M}_P^\cap$ by $\sigma = \sigma_1 \cup \sigma_2$ and thus we obtain $p\sigma = t$.

- If $p = \{q\}_r$, then since $\Gamma \vdash p : \pi$, we have

\[
\begin{align*}
\pi &= \{\|\pi_1\|\}_{\pi_2} \\
\Gamma &\vdash q : \pi_1 \\
\Gamma &\vdash r : \pi_2
\end{align*}
\]

(40)

Since $\pi\theta = \tau$, by (40) we have

\[
\begin{align*}
\tau &= \{\|\tau_1\|\}_{\tau_2} \\
\tau_1 &= \pi_1\theta \\
\tau_2 &= \pi_2\theta
\end{align*}
\]

(41)

Since $\Gamma \vdash t : \tau$ and $t$ is composed, by (41) we have

\[
\begin{align*}
t &= \{\|t_1\|\}_{t_2} \\
\Gamma_p &\vdash t_1 : \tau_1 \\
\Gamma_p &\vdash t_2 : \tau_2
\end{align*}
\]

(42)

We consider two cases:

- If $q$ is a pair, then by assumption, $q$ is a variable. Moreover, $r$ is also a variable. Since $p$ is linear, $q \neq r$. Hence we define $\sigma = [q/t_1, r/t_2]$ and thus $p\sigma = t$.

- If $q$ is a pair, then by (40) $\pi_1$ is a pair. By (41), it follows that $\tau_1$ is a pair. Hence by (42), $t_1$ is either a pair or a variable. Moreover, since $P$ is splitting and $\tau_1$ is a pair, by (42), $t_1$ is not a variable. Hence $t_1$ is a pair. By induction hypothesis, there exists $\sigma' : \text{vars}(q) \rightarrow \mathcal{M}_P^\cap$ such that $q\sigma' = t_1$. Since $p$ is linear, $r \notin \text{vars}(q)$. Hence, we define $\sigma : \text{vars}(p) \rightarrow \mathcal{M}_P^\cap$ by $\sigma = \sigma'|r/t_2$. Therefore, $p\sigma = t$.

This completes the proof of the lemma.

\[\square\]

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

**Proof.** Let $t_1, t_2 \in \mathcal{M}_P^\cap$ such that $\Gamma_p \vdash t_1 : \tau$ and $\Gamma_p \vdash t_2 : \tau$. Without loss of generality, it is enough to show that $S(t_1, \tau) \subseteq S(t_2, \tau)$ and $S(t_1, \tau) \neq \emptyset$. Since
Since \( i \in \{1, \ldots, n\} \) and a substitution \( \sigma : \mathcal{V}_{ly} \rightarrow \mathcal{Y}_p \) such that

\[
\tau = \pi_i \sigma
\]

By (43) and Lemma 15, there exists \( \gamma : \text{vars}(p_i) \rightarrow \mathcal{M}_p \) such that \( t_1 = p_i \gamma \).

Since \( \text{dom}(\gamma) \cap \text{dom}(\sigma) = \emptyset \), we set \( \theta = \sigma \cup \gamma \) to obtain \( (p_i, \pi_i)\theta = (t_1, \tau) \). Hence \( S(t_1, \tau) \neq \emptyset \). It remains to show that \( S(t_1, \tau) \subseteq S(t_2, \tau) \). Since \( S(t_1, \tau) \neq \emptyset \), let \( j \in S(t_1, \tau) \). We have to show that \( j \in S(t_2, \tau) \). Since \( j \in S(t_1, \tau) \), there exists \( \theta \) such that \( (p_j, \pi_j)\theta = (t_1, \tau) \). Hence we have

\[
\pi_j \theta = \tau.
\]

Since \( \text{vars}(p) \cap \text{vars}(\pi) = \emptyset \), by (44) there exists \( \eta : \mathcal{V}_{ly} \rightarrow \mathcal{Y}_p \) such that \( \pi_j \eta = \tau \). This by Lemma 15 implies that there is \( \xi : \text{vars}(p) \rightarrow \mathcal{M}_p \) such that \( t_2 \xi = p_j \).

Since \( \text{dom}(\eta) \cap \text{dom}(\xi) = \emptyset \), we set \( \rho = \eta \cup \xi \) to achieve \( (p_j, \pi_j)\rho = (t_2, \tau) \). Hence we have \( j \in S(t_2, \tau) \). This completes the proof of the lemma.

\[\square\]

**Corollary 1.** Let \( P \) be a splitting protocol and \( S_f = (T_f, \Gamma_f, E_f) \), where \( E_f = \{ f(p_1 : \pi_1) = u_1, \ldots, f(p_n : \pi_n) = u_n \} \), be a type-based message transformation with respect to \( \Gamma_f \). Then

\[
\forall \tau. \exists i \in \{1, \ldots, n\}. \forall t \in \mathcal{M}_p^\mathcal{Y}. t \text{ is composed } \land \Gamma_p \vdash t : \tau \Rightarrow (\exists \theta. (p_i, \pi_i)\theta = (t, \tau)) \land (\forall j < i. \forall \theta'. (p_j, \pi_j)\theta' \neq (t, \tau))
\]

**Proof.** Let \( t \in \mathcal{M}_p^\mathcal{Y} \) be a composed term such that \( \Gamma_p \vdash t : \tau \) and \( S(t, \tau) = \{ i | \exists \theta. (p_i, \pi_i)\theta = (t, \tau) \} \). By Lemma 3, we have \( S(t, \tau) \neq \emptyset \). Hence there exists the smallest \( i \in S(t, \tau) \) and a substitution \( \theta \) such that

\[
(p_i, \pi_i)\theta = (t, \tau)
\]

By Lemma 3 we have \( S(t, \tau) = S(u, \tau) \) for any term \( u \) such that \( \Gamma_p \vdash u : \tau \). Hence \( i \) is the smallest index satisfying (45) for all such \( u \). This completes the proof of the corollary.

\[\square\]

**B.2 Proof of Lemma 2 and Proposition 2**

We first prove some auxiliary results.

**Lemma 16.** Let \( P \) be a protocol and \( t, u \in \mathcal{M}_p^\mathcal{Y} \) such that \( \Gamma_p \vdash t : \tau_t \) and \( \Gamma_p \vdash u : \tau_u \). If \( \text{split}(t) \subseteq \text{split}(u) \) then \( \text{split}(\tau_t) \subseteq \text{split}(\tau_u) \).

**Corollary 2.** Let \( P \) be a protocol and \( t, u \in \mathcal{M}_p^\mathcal{Y} \) such that \( \Gamma_p \vdash t : \tau_t \) and \( \Gamma_p \vdash u : \tau_u \). If \( \text{split}(t) = \text{split}(u) \) then \( \text{split}(\tau_t) = \text{split}(\tau_u) \).

**Lemma 17.** Let \( T_f \) be a set of ground atomic and hash types. Then for all ground types \( \tau \) it holds that \( \tau \in \text{rem}(T_f) \iff \forall \tau' \in \text{set}(\text{split}(\tau)) \cdot \tau' \in \text{rem}(T_f) \).

**Proof.** We prove the lemma by induction on \( \tau \).
– If \( \tau \) is not a pair then \( \text{set}(\text{split}(\tau)) = \{\tau\} \). Hence the lemma holds trivially.
– If \( \tau = \langle \tau_1, \tau_2 \rangle \) then we have
\[
\text{set}(\text{split}(\tau)) = \text{set}(\text{split}(\tau_1)) \cup \text{set}(\text{split}(\tau_2)). \quad (46)
\]
Since \( \tau \notin T_f \), by Definition 14, we have
\[
\tau \in \text{rem}(T_f) \iff \tau_1, \tau_2 \in \text{rem}(T_f). \quad (47)
\]
Moreover, by induction hypothesis we have
\[
\tau_1 \in \text{rem}(T_f) \iff \forall \tau' \in \text{set}(\text{split}(\tau_1)), \tau' \in \text{rem}(T_f),
\tau_2 \in \text{rem}(T_f) \iff \forall \tau' \in \text{set}(\text{split}(\tau_2)), \tau' \in \text{rem}(T_f). \quad (48)
\]
By (47), (48) and (46) we obtain
\[
\tau \in \text{rem}(T_f) \iff \forall \tau' \in \text{set}(\text{split}(\tau)), \tau' \in \text{rem}(T_f).
\]
This completes the proof of the lemma. \( \square \)

**Lemma 18.** For all terms \( t \in T \) and all substitutions \( \theta \), it holds that
\[
\text{split}(t\theta) = \text{split}(\text{split}(t)\theta)
\]

**Proof.** We prove the lemma by induction on \( t \).
– If \( t \) is not a pair, then \( \text{split}(t) = \{t\} \). Hence \( \text{split}(t\theta) = \text{split}(\text{split}(t)\theta) \).
– If \( t = \langle t_1, t_2 \rangle \) then we have
\[
\text{split}(t\theta) = \text{split}(\langle t_1\theta, t_2\theta \rangle)
= \text{split}(t_1\theta) \sqcup \text{split}(t_2\theta) \quad (49)
\]
By induction hypothesis, we have
\[
\text{split}(t_1\theta) = \text{split}(\text{split}(t_1)\theta)
\text{split}(t_2\theta) = \text{split}(\text{split}(t_2)\theta) \quad (50)
\]
Moreover, we have
\[
\text{split}(t) = \text{split}(t_1) \sqcup \text{split}(t_2) \quad (51)
\]
By (49), (50) and (51) we have
\[
\text{split}(t\theta) = \text{split}(\text{split}(t_1)\theta) \cup \text{split}(\text{split}(t_2)\theta)
= \text{split}(\text{split}(t_1)\theta) \sqcup \text{split}(\text{split}(t_2)\theta)
= \text{split}(\text{split}(t)\theta)
\]
This completes the proof of the lemma. \( \square \)

**Lemma 19.** If \( T, U \) are multisets of terms such that \( T \sqsubseteq U \) and \( \theta \) is a substitution then the following holds:
Proof. We first prove that \( \text{split}(T) \subseteq \text{split}(U) \). We consider \( T \) and \( U \) are functions from terms to integers. We need to show that \( \text{split}(T)(t) \leq \text{split}(U)(t) \) for all terms \( t \in T \).

Let \( t \in T \). Suppose \( T = \{t_1, \ldots, t_n\} \) and \( U = \{t_1, \ldots, t_m\} \) such that \( n \leq m \). Then \( \text{split}(T)(t) = \sum_{1 \leq i \leq n} \text{split}(t_i)(t) \leq \sum_{1 \leq i \leq m} \text{split}(t_i)(t) = \text{split}(U)(t) \). Hence we have \( \text{split}(T) \subseteq \text{split}(U) \).

Similarly, we have
\[
T(\theta)(t) = \sum \{T(t_i) \mid 1 \leq i \leq n, t_i \in \theta = t\} \leq \sum \{T(t_i) \mid 1 \leq i \leq m, t_i \in \theta = t\} = U(\theta)(t)
\]
Therefore, we have \( \text{split}(T)(\theta) \subseteq \text{split}(U)(\theta) \).

\( \Box \)

**Lemma 20.** For all \( t, u \in T \) and all substitutions \( \theta \), \( \text{split}(t) \subseteq \text{split}(u) \) implies that \( \text{split}(t\theta) \subseteq \text{split}(u\theta) \).

**Proof.** By Lemma 18, we have \( \text{split}(t\theta) = \text{split}(\text{split}(t)\theta) \). Moreover, we have \( \text{split}(t) \subseteq \text{split}(u) \). Together with Lemma 19 we have
\[
\text{split}(t\theta) \subseteq \text{split}(\text{split}(u)\theta) = \text{split}(u\theta)
\]
This completes the proof of the lemma. \( \Box \)

**Definition 23 (Multiset of variables).** We define the multiset of variables \( VC(t) \) of a term \( t \) as follows.

- if \( t \) is a variable then \( VC(t) = \{t\} \),
- if \( t \) is an atom then \( VC(t) = \emptyset \),
- if \( t = h(u) \) then \( VC(t) = VC(u) \),
- if \( t = (t_1, t_2) \) or \( \{t_1\} \cup t_2 \), then \( VC(t) = VC(t_1) \cup VC(t_2) \).

By the same argument as we have in the proof of Lemma 20, the two following lemmas are proved immediately.

**Lemma 21.** For all terms \( t, u \in T \), if \( \text{split}(t) \subseteq \text{split}(u) \) then \( VC(t) \subseteq VC(u) \).

**Lemma 22.** For all substitutions \( \theta \) and terms \( t, u \in T \) such that \( VC(t) \subseteq VC(u) \), we have

- if \( |t| < |u| \) then \( |t\theta| < |u\theta| \), and
- if \( |t| \leq |u| \) then \( |t\theta| \leq |u\theta| \).

**Corollary 3.** For all substitutions \( \theta \) and terms \( t, u \in T \) such that \( \text{split}(t) \subseteq \text{split}(u) \), we have

1. if \( |t| < |u| \) then \( |t\theta| < |u\theta| \), and
2. if \( |t| \leq |u| \) then \( |t\theta| \leq |u\theta| \).
Lemma 23. Let \( P \) be a splitting protocol and \( S_f = (T_f, \Gamma_f, E_f) \), where \( E_f = \{ f(p_1 : \pi_1) = u_1, \ldots, f(p_n : \pi_n) = u_n \} \), be a type-based message transformation with respect to \( \Gamma_f \). Then for all terms \( t \in \text{nf}(M^\pi_f) \) with \( \Gamma_P \vdash t : \tau \), the following holds:

1. If \( t \) is a variable or an atom, then \( f(t) = t \) or \( f(t) = \text{nil} \). Moreover, if \( t \) is an asymmetric key then \( f(t^{-1}) = \text{nil} \) if and only if \( f(t) = \text{nil} \).
2. If \( t = h(u) \), then \( \tau \in \text{rem}(T_f) \) and \( f(t) = \text{nil} \) or \( f(t) = \text{nf}(h^a(f(u))) \) for some \( a \geq 0 \).
3. If \( t = t_1 \ldots t_m \), then \( f(t) = \text{nf}(\langle f(t_1) \rangle, \ldots, f(t_m)) \) for some terms \( t_i \) with \( i \in \{1, \ldots, m\} \) such that \( \text{split}(t_1, \ldots, t_m) = \text{split}(t) \), and \( |t_i| < |t| \) for all \( i \in \{1, \ldots, m\} \).
4. If \( t = \langle w \rangle^k \), then \( f(t) = \text{nf}(\langle \langle f(t_1) \rangle, \ldots, \langle f(t_m) \rangle \rangle^k) \) for some \( t_i \) and \( 1 \leq i \leq m \) such that \( \text{split}(t_1, \ldots, t_m) = \text{split}(w) \), \( b \geq 0 \), and \( |t_i| \leq |w| \) and \( a_i \geq 0 \) for all \( i \in \{1, \ldots, m\} \).

Proof. If \( t \) is a variable or an atom, then by Program 1, we have \( f(t) = t \) or \( f(t) = \text{nil} \). Moreover, if \( t \) is an asymmetric key then since \( pk(\alpha) \in T_f \) if and only if \( \text{pri}(\alpha) \in T_f \), we have \( f(t) = t \). Thus \( f(t^{-1}) \neq \text{nil} \) and \( f(t) \neq \text{nil} \). Hence the lemma holds for this case.

Now we consider the case that \( t \) is composed. Since \( \Gamma_P \vdash t : \tau \), by Corollary 1, there exists the smallest \( i \in \{1, \ldots, n\} \) and a substitution \( \theta \) such that \( (p_i, \pi_i)\theta = (t, \tau) \). Hence by Program 1, we have \( f(t) = \text{nf}(u_i\theta) \). We consider the following cases:

- If \( t = h(w) \) then we have two possibilities:
  - if \( \tau \in \text{rem}(T_f) \) then by Program 1 we obtain \( f(t) = \text{nil} \) as required, since \( \text{nil} \in IK_0 \) and \( \text{nil} \) is not an asymmetric key,
  - if \( \tau \notin \text{rem}(T_f) \) then since \( p_i \) is not a pattern variable and \( t = p_i\theta \), we must have \( p_i = h(q) \) and \( q\theta = w \). By Definition 13, we have \( u_i = h^a(f(q)) \) for some \( a \geq 0 \). Thus, we have \( f(t) = \text{nf}(h^a(f(q))) = \text{nf}(h^a(f(w))) \) as required.
- If \( t = \langle w_1, w_2 \rangle \) then since \( p_i \) is not a pattern-variable and \( t = p_i\theta \), we have \( p_i = \langle q_1, q_2 \rangle \) with
  \[
  \begin{align*}
  w_1 &= q_1\theta \\
  w_2 &= q_2\theta
  \end{align*}
  \]

By Definition 13, we have
\[
\begin{align*}
  u_i &= \langle f(t_1), \ldots, f(t_m) \rangle \\
  \text{split}(t_1, \ldots, t_m) &= \text{split}(p_i) \\
  |t_j| &= |p_i| \quad \text{for all } j \in \{1, \ldots, m\}
\end{align*}
\] (53)

and therefore the following points are trivial.

1. \( \text{split}(\langle t_1\theta, \ldots, t_m\theta \rangle) = \text{split}(t) \).
   This follows from (53) and Lemma 20.
2. \( |t_j\theta| < |t| \) for all \( j \in \{1, \ldots, m\} \).
   Let \( j \in \{1, \ldots, m\} \). From (53), we get \( \text{split}(t_j) \subseteq \text{split}(p_i) \) and \( |t_j| < |p_i| \).
   Using Corollary 3, we derive \( |t_j\theta| < |p_i\theta| = |t| \) as required.
It remains to show that \( f(t) = nf(\langle f(t_1, \theta), \ldots, f(t_m, \theta) \rangle) \). We consider two cases:

- If \( \tau \in rem(T_f) \) then by Program 1 we have \( f(t) = \text{nil} \). Let \( \Gamma_P \vdash t_j : \tau_j \) for all \( j \in \{1, \ldots, m\} \). By Point 1 and Corollary 2 we have

\[
\text{split}(\langle \tau_1, \ldots, \tau_m \rangle) = \text{split}(\tau) \quad (54)
\]

Since \( \tau \in rem(T_f) \), by (54) and Lemma 17 we have \( \tau_j \in rem(T_f) \) for all \( j \in \{1, \ldots, m\} \). Hence by Program 1 we have \( f(t, \theta) = \text{nil} \) for all \( j \in \{1, \ldots, m\} \). That means \( f(t) = nf(\langle f(t_1, \theta), \ldots, f(t_m, \theta) \rangle) \) as required.

- If \( \tau \notin rem(T_f) \) then by Program 1 we have \( f(t) = nf(\langle f(t_1, \theta), \ldots, f(t_m, \theta) \rangle) \).

  - If \( t = \{\{w\}\}_k \) then since \( p_i \) is not a pattern-variable and \( t = p_i, \theta \), we must have \( p_i = \{q\}_r \) with

\[
w = q\theta \\
k = r\theta
\]

From Definition 13, we have

\[
u_i = \{\{\langle f(t_1)\rangle_{f(\tau_1)}^\alpha, \ldots, \{f(t_m)\}_{f(\tau_m)}^\alpha\}\}_{f(\tau)^\beta}
\]

\[
\text{split}(\langle t_1, \ldots, t_m \rangle) = \text{split}(q)
\]

\[
|t_j| \leq |q| \text{ for all } j \in \{1, \ldots, m\}
\]

and therefore the following two points hold.

1. \( \text{split}(\langle t_1, \ldots, t_m \rangle) = \text{split}(w) \).

   This point follows from (55), (56) and Lemma 20.

2. \( |t_j, \theta| \leq |w| \) for all \( j \in \{1, \ldots, m\} \).

   Let \( j \in \{1, \ldots, m\} \). From (56) we have \( \text{split}(t_j) \subseteq \text{split}(q) \) and \( |t_j| \leq |q| \).

   Using Corollary 3, we derive \( |t_j, \theta| \leq |q, \theta| = |w| \) as required.

It remains to show that

\[
f(t) = nf(\{\{\langle f(t_1, \theta)\rangle_{f(\tau_1)}^\alpha, \ldots, \{f(t_m, \theta)\}_{f(\tau_m)}^\alpha\}\}_{f(\tau)^\beta}). \quad (57)
\]

Let \( \Gamma_P \vdash w : \tau' \). We consider two cases:

- If \( \tau \in rem(T_f) \) then by Program 1 we have \( f(t) = \text{nil} \). Moreover, since \( \tau \notin T_f \) we must have \( \tau' \in rem(T_f) \). By a similar reasoning as before, we also have

\[
f(t, \theta) = \text{nil} \text{ for all } j \in \{1, \ldots, m\}.
\]

   Hence we obtain (57).

- If \( \tau \notin rem(T_f) \) then by Program 1 we have

\[
f(t) = nf(\{\{\langle f(t_1, \theta)\rangle_{f(\tau_1)}^\alpha, \ldots, \{f(t_m, \theta)\}_{f(\tau_m)}^\alpha\}\}_{f(\tau)^\beta}).
\]

   By (55), this yields (57)

This completes the proof of the lemma. \( \square \)

**Lemma (Justification of Lemma 2).** Let \( P \) be a splitting protocol and \( S_f = (T_f, T_j, E_f) \) be a type-based message transformation. Suppose \( t \in nf(M^{\alpha}) \setminus \{\text{nil}\} \) and \( \Gamma_P \vdash t : \tau \). Then \( \tau \in rem(T_f) \) iff \( f(t) = \text{nil} \).
Proof. Suppose \( t \in \mathsf{nf}(\mathcal{M}^t_p) \setminus \{\text{nil}\} \) and \( \Gamma_P \vdash t : \tau \). By Program 1 we have \( \tau \in \mathsf{rem}(T_f) \) implies \( f(t) = \text{nil} \). It remains to show that \( f(t) = \text{nil} \) implies \( \tau \in \mathsf{rem}(T_f) \). We prove this by induction on the size of \( t \).

- Base case: \( t \) is a variable or an atom. Then by Program 1, we have \( f(t) = \text{nil} \) implies \( \tau \in \mathsf{rem}(T_f) \) or \( t = \text{nil} \). Since \( t \neq \text{nil} \), we must have \( \tau \in \mathsf{rem}(T_f) \).
- Inductive cases: Suppose that \( f(t) = \text{nil} \). We need to show that \( \tau \in \mathsf{rem}(T_f) \).
  
  - If \( t = h(u) \) then assume by contradiction that \( \tau \notin \mathsf{rem}(T_f) \). By Lemma 23, we have \( f(t) = \mathsf{nf}(h^a(f(u))) \) for some \( a \geq 0 \). Since \( f(t) = \text{nil} \), we also have \( f(u) = \text{nil} \). Let \( \Gamma_P \vdash u : \tau_u \). Since \( \Gamma_P \vdash t : \tau \), we have \( \tau = h(\tau_u) \).
    
    - Since \( t \in \mathsf{nf}(\mathcal{M}^t_p) \setminus \{\text{nil}\} \), we also have \( u \in \mathsf{nf}(\mathcal{M}^t_p) \setminus \{\text{nil}\} \). Hence by induction hypothesis and \( f(u) = \text{nil} \), we have \( \tau_u \in \mathsf{rem}(T_f) \). This by Definition 14 implies that \( \tau \in \mathsf{rem}(T_f) \) which is a contradiction. Therefore we must have \( \tau \in \mathsf{rem}(T_f) \) as required.
  
  - If \( t = \langle u_1, u_2 \rangle \) then by Lemma 23, we have
    
    \[
    f(t) = \mathsf{nf}(\langle f(t_1), \ldots, f(t_m) \rangle) \\
    \mathsf{split}(\langle t_1, \ldots, t_m \rangle) = \mathsf{split}(t) \\
    |t_i| < |t| \text{ for all } i \in \{1, \ldots, m\}
    \]
    
    by (58) and Corollary 2 we have
    
    \[
    \mathsf{split}(\tau) = \mathsf{split}(\langle \tau_1, \ldots, \tau_m \rangle)
    \]
    
    Suppose that \( f(t) = \text{nil} \). We need to show that \( \tau \in \mathsf{rem}(T_f) \). Since \( f(t) = \text{nil} \), by (58) we have \( f(t_i) = \text{nil} \) for all \( i \in \{1, \ldots, m\} \). Moreover, since \( t \in \mathsf{nf}(\mathcal{M}^t_p) \setminus \{\text{nil}\} \), we also have \( t_i \in \mathsf{nf}(\mathcal{M}^t_p) \setminus \{\text{nil}\} \) for all \( i \in \{1, \ldots, m\} \). Hence for all \( i \in \{1, \ldots, m\} \), since \( |t_i| < |t| \), by induction hypothesis and \( f(t_i) = \text{nil} \), we have \( \tau_i \in \mathsf{rem}(T_f) \). This together with (59) and Lemma 17 yield that \( \tau \in \mathsf{rem}(T_f) \).
  
  - If \( t = \{u\}_k \) then by Lemma 23, we have
    
    \[
    f(t) = \mathsf{nf}(\langle \{f(t_1)\}_1, \ldots, \{f(t_m)\}_1 \rangle) \\
    \mathsf{split}(\{t_1, \ldots, t_m\}) = \mathsf{split}(u) \\
    |t_i| \leq |u| \text{ for all } i \in \{1, \ldots, m\}
    \]
    
    Suppose that \( f(t) = \text{nil} \). We need to show that \( \tau \in \mathsf{rem}(T_f) \). Since \( f(t) = \text{nil} \), by (60) we have \( f(t_i) = \text{nil} \) for all \( i \in \{1, \ldots, m\} \). Moreover, since \( t \in \mathsf{nf}(\mathcal{M}^t_p) \setminus \{\text{nil}\} \), we also have \( t_i \in \mathsf{nf}(\mathcal{M}^t_p) \setminus \{\text{nil}\} \) for all \( i \in \{1, \ldots, m\} \). Moreover, since \( |t_i| \leq |u| < |t| \) and \( f(t_i) = \text{nil} \) for all \( i \in \{1, \ldots, m\} \), by induction hypothesis, we have \( \tau_i \in \mathsf{rem}(T_f) \) for all \( i \in \{1, \ldots, m\} \). This together with (60), Corollary 2 and Lemma 17 yield that \( \tau_u \in \mathsf{rem}(T_f) \). Moreover, by Definition 14, \( \tau_u \in \mathsf{rem}(T_f) \) implies \( \tau \in \mathsf{rem}(T_f) \). Hence we have \( \tau \in \mathsf{rem}(T_f) \) as required.

This completes the proof of the lemma. □
**Proposition 5.** Let $P$ be a splitting protocol and $S_f$ be a type-based protocol transformation. Then the function $f$ defined by Program 1 terminates on all terms $t \in \mathcal{M}_P^2$.

**Proof.** Let $S_f = (E_f, \Gamma_f, \mathcal{E}_f)$. It is clear that the normal form function $nf$ terminates. Therefore, it remains to show that $f^{rec}$ terminates on all normal form terms $t$ in $\mathcal{M}_P^2$. We prove this by induction on the size of $t$. Let $\Gamma_P \vdash t : \tau$. If $t$ is a variable or an atom then the termination of $f^{rec}(t)$ is immediate. For composed messages, we distinguish two cases. If $\tau \in \text{rem}(T_f)$ then $f^{rec}$ terminates with output $\text{nil}$. Otherwise, by an inspection of Definition 13 and Program 1 we see that $f^{rec}$ is called recursively on normal form terms smaller than $t$. Those calls terminate by the induction hypothesis from which the termination of $f^{rec}(t)$ follows immediately. Hence, we have shown that $f$ terminates on all inputs $t \in \mathcal{M}_P^2$ as required. \hfill \Box

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

**Proof.** The proposition follows from the following three points. (i) $f$ is a total function, i.e., it terminates on all inputs $t \in \mathcal{M}_P^2$. This follows from Proposition 5. (ii) Conditions (1)-(5) of Definition 6 are satisfied. This immediately follows from Lemma 23. (iii) For each set of terms $T \in \{\mathcal{M}_P, \mathcal{M}_P^2, \mathcal{N}_P\}$, $f$ transforms terms in $T$ to terms in $T$. This holds because $\text{atoms}(f(t)) \cup \text{vars}(f(t)) \subseteq \text{atoms}(t) \cup \text{vars}(t) \cup \{\text{nil}\} \subseteq T$ and $T$ is closed under message composition. \hfill \Box
Proof. We prove the lemma by induction on Lemma 24.

Definition 24 (nil-free substitutions). A substitution θ is nil-free with respect to a set of variables V iff for all variables x ∈ dom(θ) ∩ V it holds that nil ∉ St(xθ).

Lemma 24. Let t ∈ T and θ be a substitution. If nf(t) = nil, then nf(tθ) = nil.

Proof. We prove the lemma by induction on t.

− If t is a variable or an atom, then nf(t) = nil implies that t = nil. Hence nf(tθ) = nil.
− If t = h(u), then nf(t) = nil implies that nf(u) = nil. Hence by induction hypothesis, we have nf(uθ) = nil. Thus nf(tθ) = nf(h(uθ)) = nil.
− If t = ⟨t1,t2⟩, then nf(t) = nil implies that nf(t1) = nf(t2) = nil. Hence by induction hypothesis, we have nf(t1θ) = nf(t2θ) = nil. Thus nf(tθ) = nf(⟨t1θ, t2θ⟩) = nil.
− If t = [[u]]k, then nf(t) = nil implies that nf(u) = nil. Hence by induction hypothesis, we have nf(uθ) = nil. Thus nf(tθ) = nf([[uθ]]kθ) = nil.

This completes the proof of the lemma. □

Lemma 25. Let θ be a nil-free substitution with respect to V. Then for all terms t ∈ T such that vars(t) ⊆ V, we have nf(tθ) = nf(tθ).

Proof. We prove the lemma by induction on t.

− If t is a variable or an atom, then nf(t) = t. Hence nf(tθ) = tθ. Moreover, since θ is nil-free with respect to V and vars(t) ⊆ V, we have nf(tθ) = tθ. Therefore, we have nf(tθ) = nf(tθ).
− If t = h(u), then by Definition 5, either nf(t) = nil or nf(t) = h(nf(u)).
  • If nf(t) = nil, then nf(u) = nil. Hence by Lemma 24, we have nf(uθ) = nil. Hence nf(tθ) = nf(h(uθ)) = nil. Moreover, we have nf(tθ) = nil. Thus nf(tθ) = nf(tθ).
  • If nf(t) = h(nf(u)), then nf(tθ) = h(nf(u)θ) = h(nf(u)θ). Moreover, we have nf(tθ) = nf(h(uθ)). By induction hypothesis, we have nf(uθ) = nf(uθ). Thus nf(tθ) = h(nf(uθ)). Hence nf(tθ) = nf(tθ).
− If t = ⟨t1,t2⟩, then by induction hypothesis, we have
  \[ nf(t1θ) = nf(t1θ) \]
  \[ nf(t2θ) = nf(t2θ) \]  \hspace{1cm} (61)

Moreover, we have
\[ nf(tθ) = nf(⟨t1θ,t2θ⟩) \]  \hspace{1cm} (62)

By Definition 5, we have three cases:
• If \( nf(t_1) = \text{nil} \) then \( nf(t) = nf(t_2) \). Hence \( nf(t)\theta = nf(t_2)\theta \). This together with (61) yield

\[ nf(t)\theta = nf(t_2)\theta \]  

(63)

Moreover, by Lemma 24, we have \( nf(t_1\theta) = \text{nil} \). This together with (62) imply that

\[ nf(t\theta) = nf(t_2\theta) \]  

(64)

By (63) and (64), we have \( nf(t)\theta = nf(t\theta) \).

• If \( nf(t_2) = \text{nil} \) then similarly as above, we obtain \( nf(t)\theta = nf(t\theta) \).

• If \( nf(t_1) \neq \text{nil} \) and \( nf(t_2) \neq \text{nil} \), then \( nf(t) = \langle nf(t_1), nf(t_2) \rangle \). Hence we have

\[ nf(t)\theta = \langle nf(t_1)\theta, nf(t_2)\theta \rangle \]  

(65)

Moreover, since \( \theta \) is nil-free with respect to \( V \) and \( \text{vars}(t) \subseteq V \), by (61) it implies that \( nf(t_1\theta) \neq \text{nil} \) and \( nf(t_2\theta) \neq \text{nil} \). This by (62) and Definition 5 yields

\[ nf(t\theta) = \langle nf(t_1\theta), nf(t_2)\theta \rangle \]  

(66)

By (61), (65) and (66) we have \( nf(t)\theta = nf(t\theta) \).

– If \( t = \{u\}_k \), then similarly as above, we have \( nf(t)\theta = nf(t\theta) \).

This completes the proof of the lemma.

\[ \square \]

**Lemma 26.** Let \( P \) be a splitting protocol and \( S_f \) be a type-based protocol transformation and \( \theta \) be a well-typed substitution with respect to \( \Gamma_P \). Then \( f(\theta) \) is nil-free with respect to \( \mathcal{V}_P \).

**Proof.** Since \( \theta \) is well-typed with respect to \( \Gamma_P \) and \( \mathcal{C}_P \) does not contain nil, \( \theta \) is nil-free with respect to \( \mathcal{V}_P \). Let \( x \in \text{dom}(f(\theta)) \cap \mathcal{V}_P \). We consider two cases:

– If \( f(x\theta) = \text{nil} \) then since \( \theta \) is nil-free with respect to \( \mathcal{V}_P \), we have

\[ x\theta \in nf(\mathcal{M}_P^f) \setminus \{\text{nil}\} \]  

(67)

Let \( \Gamma_P \vdash x : \tau \). Since \( \theta \) is well-typed with respect to \( \Gamma_P \), we must have \( \Gamma_P \vdash x\theta : \tau \). Since (67) and \( x \in nf(\mathcal{M}_P^f) \setminus \{\text{nil}\} \), by applying Lemma 2 twice, we deduce \( f(x) = \text{nil} \) from \( f(x\theta) = \text{nil} \). Hence, \( x \notin \text{dom}(f(\theta)) \), which contradicts our assumption.

– If \( f(x\theta) \neq \text{nil} \) then nil \( \notin \text{St}(f(x\theta)) \) since \( f(x\theta) \) is in normal form. Hence, we have \( \text{nil} \notin \text{St}(xf(\theta)) \).

We have established that nil \( \notin \text{St}(xf(\theta)) \) for all \( x \in \text{dom}(f(\theta)) \cap \mathcal{V}_P \), i.e., \( f(\theta) \) is nil-free with respect to \( \mathcal{V}_P \) as required.

\[ \square \]

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

\[ f(t\theta) = f(t)f(\theta) \]
Proof. To reduce notational clutter, we rename the function \( f \) to \( f_0 \) and \( f_{\text{rec}} \) to \( f \) in this proof. Hence, we have to prove that \( f_0(t\theta) = f_0(t)f_0(\theta) \). From the definition of \( f_0 \) (function \( f \) in Program 1) and Lemmas 26 and 25, we deduce \( f_0(\theta) = f(\theta) \) and

\[
f_0(t\theta) = f(nf(t\theta)) = f(nf(t)\theta) \quad f_0(t)f_0(\theta) = f(nf(t))f(\theta).
\]

Therefore, it is sufficient to prove the statement \( f(t\theta) = f(t)f(\theta) \) for all terms \( t \) in normal form. We do this by induction on the recursion structure of the function \( f \) (i.e., \( f_{\text{rec}} \) in Program 1). This induction scheme is well-defined, since the program terminates by Proposition 5. Let \( \tau = \Gamma_p(t) \), i.e., \( \Gamma_p \vdash t : \tau \).

We distinguish three base cases:

- \( \tau \in \text{rem}(T_f) \). Since \( \theta \) is well-typed with respect to \( \Gamma_p \), by Lemma 4 we have \( \Gamma_p \vdash t\theta : \tau \). By Program 1 we have \( f(t) = \text{nil} \) and \( f(t\theta) = \text{nil} \). Hence, \( f(t)f(\theta) = f(t) \).
- \( t \equiv a \) is an atom and \( \tau \notin \text{rem}(T_f) \). By Program 1 we have \( f(a) = a \). Hence, we obtain \( f(a\theta) = f(a) = a = a f(\theta) = f(a)f(\theta) \).
- \( t \equiv X \) is a variable and \( \tau \notin \text{rem}(T_f) \). By Program 1 we have \( f(X) = X \). We distinguish two cases:
  - If \( X \in \text{dom}(\theta) \), then \( f(X)f(\theta) = X f(\theta) = f(X) \).
  - 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, \( f(X\theta) = f(X) = f(X)f(\theta) \).

We prove the inductive cases after three preparatory remarks, which are common to all of them. First, note that since \( t \) is in normal form and recursive calls of \( f \) have subterms of \( t \) as argument, these subterms are also in normal form. This is needed to apply the induction hypotheses below. Second, since \( \theta \) is well-typed with respect to \( \Gamma_p \), \( f(\theta) \) is nil-free with respect to \( \Pi_\theta \) by Lemma 26. Since for all terms \( t \) we consider below, we have \( \text{vars}(t) \subseteq \Pi_\theta \), the precondition of Lemma 25 is satisfied. Third, by Corollary 1, there exists the smallest \( i \in \{1, \ldots, n\} \) and a substitution \( \theta' \) such that \( (p_i, \pi_i)\theta' = (t, \tau) \). Hence \( (p_i, \pi_i)\theta \circ \theta' = (t\theta, \tau) \). By Corollary 1, \( i \) is also the smallest such that \( (p_i, \pi_i) \) matches \( (t\theta, \tau) \). Hence, by Program 1 (modulo renamings), we have

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

We consider the following cases:

- If \( t \equiv h(v) \) then we have \( p_i = h(q), \quad v = q\theta' \), and \( u_i = h^a(f(q)) \) for some \( a \geq 0 \). Hence, using Lemma 25, we calculate

\[
f(t)f(\theta) = nf(h^a(f(v)))f(\theta) = nf(h^a(f(v))f(\theta)) = nf(h^a(f(v)f(\theta)))
\]

and

\[
f(t\theta) = nf(h^a(f(v\theta))).
\]

The desired result follows by the induction hypothesis: \( f(v\theta) = f(v)f(\theta) \).
Lemma 27. Let $P$ be a splitting protocol, $S_f$ be a type-based message transformation. Then we have, for all $t \in \text{nf}(\mathcal{M}_P^2) \setminus \{\text{nil}\}$,

$$\text{acc}(t) \cap \{X \in \mathcal{V}_P \mid f(X) = X\} \subseteq \text{acc}(f(t)) \cap \mathcal{V}_P.$$ 

Proof. We prove the lemma by induction on the size of $t$. We use the abbreviation $V_f = \{X \in \mathcal{V}_P \mid f(X) = X\}$. 

This completes the proof of the theorem. 

C.2 Well-definedness of protocol transformations

We show that if $P$ is a protocol then so is $f(P)$. The proof uses the following lemma about accessible variables.

Lemma 25 to derive

$$f(t)f(\theta) = nf((f(t_1), \ldots, f(t_m)))f(\theta)$$

$$= nf((f(t_1), \ldots, f(t_m))f(\theta))$$

and

$$= nf((f(t_1)f(\theta), \ldots, f(t_m)f(\theta)))$$

Since, by Corollary 3, we have $|t_i| < |t|$, the induction hypotheses are $f(t_i) = f(t_i)f(\theta)$ for all $i \in \{1, \ldots, m\}$. This proves the case for pairs.

- If $t = \langle w_1, w_2 \rangle$ then we have $p_i = \langle q_1, q_2 \rangle$, $w_1 = q_1\theta'$, $w_2 = q_2\theta'$, and $u_i = \{f(r_1), \ldots, f(r_m)\}$ for split$(\langle q_1, q_2 \rangle) = \text{split}(\langle r_1, \ldots, r_m \rangle)$ and $|r_i| < |\langle q_1, q_2 \rangle|$ for all $i \in \{1, \ldots, m\}$. Using the abbreviations $t_i = r_i\theta'$ and Lemma 25, we calculate

$$f(t)f(\theta) = nf((f(t_1), \ldots, f(t_m)))f(\theta)$$

$$= nf((f(t_1), \ldots, f(t_m))f(\theta))$$

and

$$= nf((f(t_1)f(\theta), \ldots, f(t_m)f(\theta)))$$

such that split$(q) = \text{split}(\langle s_1, \ldots, s_m \rangle)$, $b \geq 0$, and, for all $i \in \{1, \ldots, m\}$, $a_i \geq 0$ and $|s_i| \leq |q|$. We abbreviate $t_i = s_i\theta'$, $k' = f(k)f(\theta)$ and use Lemma 25 to derive

$$f(t)f(\theta) = nf(\langle f(t_1) \rangle f(k)\theta^a, \ldots, f(t_m) \rangle f(k)\theta^a)f(\theta)$$

$$= nf(\langle f(t_1) \rangle f(k)\theta^a, \ldots, f(t_m) \rangle f(k)\theta^a)f(\theta)$$

and

$$f(t\theta) = nf(\langle f(t_1\theta) \rangle f(k)\theta^a, \ldots, f(t_m\theta) \rangle f(k)\theta^a)$$

Since $|k| < |t|$ and, by Corollary 3, $|t_i| \leq |w| < |t|$ for all $i \in \{1, \ldots, m\}$, the induction hypotheses are $f(t_i\theta) = f(t_i)f(\theta)$ for $i \in \{1, \ldots, m\}$ and $f(k\theta) = f(k)f(\theta)$. Applying these we get the desired result.

This completes the proof of the theorem. 

□
– If \( t = Z \) is a variable then there are two cases:
  • if \( f(Z) = \text{nil} \) then \( \text{acc}(Z) \cap V_f = \emptyset \subseteq \text{acc}(f(Z)) \cap V_P \) as required,
  • if \( f(Z) = Z \) then \( \text{acc}(Z) \cap V_f = \{ Z \} = \text{acc}(f(Z)) \cap V_P \) and thus the lemma also holds.
– If \( t \) is an atom or hash then \( \text{acc}(t) \cap V_f = \emptyset \). Hence the lemma holds for this case.
– If \( t = \langle u_1, u_2 \rangle \) then we know from Lemma 23 that
  \[
  f(t) = \text{nf}(\langle f(t_1), \ldots, f(t_m) \rangle),
  \]
  \[
  \text{split}(t) = \text{split}(\langle t_1, \ldots, t_m \rangle), \text{ and}
  \]
  \[
  |t_i| < |t| \text{ for all } i \in \{1, \ldots, m\}.
  \]

Using this information we calculate as follows.

\[
\begin{align*}
\text{acc}(t) \cap V_f &= \text{acc}(\langle t_1, \ldots, t_m \rangle) \cap V_f \\
&= \bigcup_{i=1}^m (\text{acc}(t_i) \cap V_f) \\
&\subseteq \bigcup_{i=1}^m (\text{acc}(f(t_i)) \cap V_P) \quad (68) \\
&= \text{acc}(\langle f(t_1), \ldots, f(t_m) \rangle) \cap V_P \\
&= \text{acc}(\text{nf}(\langle f(t_1), \ldots, f(t_m) \rangle)) \cap V_P \quad (69) \\
&= \text{acc}(f(t)) \cap V_P \\
&= \text{acc}(\langle f(t_1), \ldots, f(t_m) \rangle) \cap V_f \\
&\subseteq \bigcup_{i=1}^m (\text{acc}(f(t_i)) \cap V_P) \\
&= \text{acc}(\langle f(t_1), \ldots, f(t_m) \rangle) \cap V_P \\
&= \text{acc}(\text{nf}(\langle f(t_1), \ldots, f(t_m) \rangle)) \cap V_P \\
&= \text{acc}(f(t)) \cap V_P \quad (70)
\end{align*}
\]

We derive (68) from the fact that \( \text{split}(\langle t_1, \ldots, t_m \rangle) = \text{split}(t) \). In (69), we apply the induction hypothesis to the terms \( t_i \). Finally, in (70), we use the fact that the set of accessible variables is invariant under normalisation.

– If \( t = \{ u \}_k \) then since \( x \in \text{acc}(t) \) then Lemma 23 informs us that
  \[
  |t_i| \leq |u| \text{ for all } i \in \{1, \ldots, m\}.
  \]

Using this information we calculate as follows.

\[
\begin{align*}
\text{acc}(t) \cap V_f &= \text{acc}(\langle t_1, \ldots, t_m \rangle) \cap V_f \\
&= \bigcup_{i=1}^m (\text{acc}(t_i) \cap V_f) \\
&\subseteq \bigcup_{i=1}^m (\text{acc}(f(t_i)) \cap V_P) \\
&= \text{acc}(\langle f(t_1), \ldots, f(t_m) \rangle) \cap V_P \\
&= \text{acc}(\text{nf}(\langle f(t_1), \ldots, f(t_m) \rangle)) \cap V_P \\
&= \text{acc}(f(t)) \cap V_P \quad (71)
\end{align*}
\]

Since \( \text{acc}(t) = \text{acc}(u) \) and \( \text{split}(u) = \text{split}(\langle t_1, \ldots, t_m \rangle) \), we obtain (71).

In (72), we apply the induction hypothesis to the terms \( t_i \). Finally, in (73), we use the fact that the set of accessible variables is invariant under normalisation and the definition of accessible terms for encryption.

This completes the proof of the lemma. ☐
**Proposition (Well-definedness; Justification of Proposition 3).** Let $P$ be a splitting protocol and $S_f$ be a type-based protocol transformation. Then $f(P)$ is a protocol with honest substitution $\delta_{f(P)} = f(\delta_P)$.

**Proof.** Let $S = [(s_1, r_1), \ldots, (s_n, r_n)]$ be the list of matching send and receive events of the protocol $P$. We define the list $f(S)$ as follows.

$$f(S) = [(f(s_{i_1}), f(r_{i_1})), \ldots, (f(s_{i_m}), f(r_{i_m}))]$$

where $\{i_1, \ldots, i_m\} = \{i \mid 1 \leq i \leq n \land f(\text{term}(s_i)) \neq \text{nil}\}$.

Since $\text{term}(s_i)$ and $\text{term}(r_i)$ have the same type with respect to $\Gamma_P$ and $\{\text{term}(s_i), \text{term}(r_i)\} \subseteq nf(M_P) \setminus \{\text{nil}\}$, by Lemma 2, we have $f(\text{term}(s_i)) \neq \text{nil}$ if and only if $f(\text{term}(r_i)) \neq \text{nil}$. This implies $f(S)$ is the list of matching send and receive events of $f(P)$. We need to establish three points for $f(P)$ to be a protocol.

1. the sets of variables and fresh values occurring in different roles are pairwise disjoint,
2. for all $f(x) \in f(P)(R)$ and $X \in \text{vars}(\text{term}(e))$, there is an event $\text{recv}(t)$ in $f(P)(R)$ such that $\text{recv}(t)$ equals or precedes $f(e)$ in $P(R)$ and $X \in \text{acc}(t)$,
3. there exists the honest substitution $\delta_{f(P)}$.

**Point 1** Since $f(X) = X$ or $f(X) = \text{nil}$ for all $X \in V_P$ and $P$ is a protocol, the first point holds trivially.

**Point 2** We assume that $f(e) \in f(P)(R)$ and $X \in \text{vars}(\text{term}(f(e)))$. Since $X \in \text{vars}(\text{term}(f(e)))$, we have $X \in \text{vars}(\text{term}(e))$. Since $P$ is a protocol, this implies that there exists $j \in \{1, \ldots, n\}$ such that

$$X \in \text{acc}(\text{term}(r_j))$$

and $r_j$ equals or precedes $e$ in $P(R)$. Since $X \in \text{acc}(\text{term}(r_j))$, by Lemma 27 we have

$$X \in \text{acc}(f(\text{term}(r_j)))$$

Hence, we have $f(\text{term}(r_j)) \neq \text{nil}$. Therefore, we have $j \in \{i_1, \ldots, i_m\}$ and $f(r_j) \in f(P)(R)$. Moreover, since $r_j$ equals or precedes $e$ in $P(R)$, $f(r_j)$ equals or precedes $f(e)$ in $f(P)(R)$. This shows that the second point holds.

**Point 3** It remains to show that $f(\sigma_P)$ is the honest substitution of $f(P)$. We abbreviate $t_i = \text{term}(s_i)$ and $u_i = \text{term}(r_i)$. Since $P$ is a protocol, the honest substitution of $P$ is $\delta_P = \delta_n \circ \cdots \circ \delta_1$ where, for all $k \in \{1, \ldots, n\}$,

$$\delta_k = \text{mgu}(t_k(\delta_{k-1} \circ \cdots \circ \delta_1), u_k(\delta_{k-1} \circ \cdots \circ \delta_1))$$

(74)

Using point 2) in the definition of protocols, we observe that

\footnote{We assume that the empty composition of functions is the identity by convention, hence we have $\delta_1 = \text{mgu}(t_1, u_1)$ as in Definition 3.}
Hence, we obtain···◦?

\(f\) is a protocol. ⊓ ⊔

as required. Therefore, point (3) also holds. This concludes the proof that \(f(P)\) is a protocol.

\(\square\)

\(\text{(O1)}\) \(t_k(\delta_{k-1} \circ \cdots \circ \delta_1)\) is ground and
\(\text{(O2)}\) \(\delta_k\) is a matcher with domain \(\text{dom}(\delta_k) = \text{acc}(u_k(\delta_{k-1} \circ \cdots \circ \delta_1)) \cap \mathcal{V}_P\).

Note that since the substitutions \(\delta_i\) are well-typed by construction, we will freely apply the Substitution Property (Theorem 2) in the remainder of this proof.

Below we will establish that the following two claims hold for all \(k \in \{1, \ldots, n\}\).

\(\text{(C1)}\) \(\text{dom}(f(\delta_k)) \subseteq \text{vars}(f(u_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1))).\)
\(\text{(C2)}\) \(f(\delta_k) = \text{mgu}(f(t_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1)), f(u_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1)))\)

It follows from Claim (C2) that, for all \(i \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_m\}\), we have \(f(t_i) = \text{nil}\) and \(f(u_i) = \text{nil}\) and hence \(f(\delta_i) = \text{id}\). Hence, \(f(P) = f(\delta_{i_m}) \circ \cdots \circ f(\delta_{i_1})\) is the honest substitution of \(f(P)\). Moreover, it follows from Theorem 2 that

\[f(\delta_P) = f(\delta_n) \circ \cdots \circ f(\delta_1) = \delta_f(P)\]

The proof of Claim (C2) relies on Claim (C1), which we prove first. Abbreviating \(V_f = \{X \in \mathcal{V}_P \mid f(X) = X\}\), we calculate as follows.

\[\text{dom}(f(\delta_k)) = \text{dom}(\delta_k) \cap V_f \]
\[= \text{acc}(u_k(\delta_{k-1} \circ \cdots \circ \delta_1)) \cap V_f \]
\[\subseteq \text{acc}(f(u_k)(\delta_{k-1} \circ \cdots \circ \delta_1)) \cap \mathcal{V}_P \]
\[= \text{acc}(f(u_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1))) \cap \mathcal{V}_P \]
\[\subseteq \text{vars}(f(u_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1)))\]

The first equation holds by definition of \(f\) applied to substitutions. The second equation holds by (O2) and the fact that \(V_f \subseteq \mathcal{V}_P\). Lemma 27 justifies the first set inclusion. The last equation is uses Theorem 2. Therefore, we have established Claim (C1).

We are now ready to establish Claim (C2). By (74) and observation (O2) we have

\[t_k(\delta_{k-1} \circ \cdots \circ \delta_1) = u_k(\delta_k \circ \delta_{k-1} \circ \cdots \circ \delta_1).\]

We apply \(f\) on both sides, followed by a repeated application of Theorem 2. This yields

\[f(t_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1)) = f(u_k)(f(\delta_k) \circ f(\delta_{k-1}) \circ \cdots \circ f(\delta_1)).\]

By (O1) the term \(f(t_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1))\) is ground. Hence, the substitution \(f(\delta_k)\) is a (ground) matcher. Together with Claim (C1) this implies

\[\text{dom}(f(\delta_k)) = \text{vars}(f(u_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1))).\]

Since any other matcher \(\theta\) for \(f(t_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1))\) and \(f(u_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1))\) satisfies \(\text{dom}(f(\delta_k)) \subseteq \theta\), it follows that \(f(\delta_k)\) is the most general one. Hence, we obtain

\[f(\delta_k) = \text{mgu}(f(t_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1)), f(u_k)(f(\delta_{k-1}) \circ \cdots \circ f(\delta_1)))\]
as required. Therefore, point (3) also holds. This concludes the proof that \(f(P)\) is a protocol.
C.3 Simulation

Lemma 28. Let $t \in \mathcal{T}$ and $\theta$ be a well-typed ground substitution with respect to a typing environment $\Gamma$. If $\Gamma \vdash t : \tau$ and $\{\tau\}$ is simple-keyed then $\{t\theta\}$ is simple-keyed.

Proof. Since $\theta$ is well-typed with respect to $\Gamma$, we have $\Gamma \vdash t : \tau$ by Lemma 4. Since both $\theta$ and $\tau$ are ground and $\{\tau\}$ is simple-keyed, an inspection of the type system suffices to see that $\{t\theta\}$ is also simple-keyed. □

Lemma (Justification of Lemma 5). If $P$ is a simple-keyed protocol, $T \subseteq R_{\mathcal{T}}^P$ and $\theta$ is well-typed ground substitution with respect to $\Gamma_P$, then $T\theta$ is a simple-keyed.

Proof. By assumption, $\Gamma_P(R_{\mathcal{T}}^P)$ is simple-keyed. Hence $\Gamma_P(R_{\mathcal{T}}^P)$ is also simple-keyed. Let $t \in T$ and $\Gamma_P \vdash t : \tau$. Since $\{\tau\} \subseteq \Gamma_P(R_{\mathcal{T}}^P)$ is simple-keyed, we use Lemma 28 to derive that $\{t\theta\}$ is also simple-keyed. Therefore, $T\theta$ is simple-keyed. □

Proposition (Justification of Proposition 4). Let $P$ be a simple-keyed, splitting protocol, $S_f$ a type-based message transformation, and $\theta$ a well-typed ground substitution with respect to $\Gamma_P$. Assume that $IK_0$ is simple-keyed and $f(IK_0) \subseteq IK_0$. Then, for all $T \subseteq R_{\mathcal{T}}^P$, and $u \in \mathcal{M}_P$, we have $T\theta \cup IK_0 \vdash u\theta$ implies $f(T)f(\theta) \cup IK_0^0 \vdash f(u)f(\theta)$.

Proof. By Lemma 5, $T\theta$ is simple-keyed. Hence $T\theta \cup IK_0$ is simple-keyed. Therefore, by Theorem 1, $T\theta \cup IK_0 \vdash u\theta$ implies $f(T\theta \cup IK_0) \cup \{\text{nil}\} \vdash f(u\theta)$. Since $f(T\theta \cup IK_0) = f(T\theta) \cup f(IK_0)$, we obtain $f(T\theta) \cup f(IK_0) \cup \{\text{nil}\} \vdash f(u\theta)$. Together with $f(IK_0) \subseteq IK_0^0$ and $\text{nil} \in IK_0^0$ this yields $f(T\theta) \cup IK_0^0 \vdash f(u\theta)$. By Theorem 2, we have $f(T\theta) = f(T)f(\theta)$ and $f(u\theta) = f(u)f(\theta)$. Hence, we have $f(T)f(\theta) \cup IK_0^0 \vdash f(u)f(\theta)$ as required. □

Lemma 29. Let $\sigma$ be a well-typed ground substitution with respect to $\Gamma_P$. Then $f(\sigma)$ is a well-typed ground substitution with respect to $\Gamma_{f(P)}$.

Proof. Let $\sigma$ be a well-typed ground substitution with respect to $\Gamma_P$. Assume that $X \in \text{dom}(f(\sigma))$ and $\Gamma_{f(P)} \vdash X : \tau$. We have to show that

1. $f(\sigma)$ is ground, and
2. $\Gamma_{f(P)} \vdash Xf(\sigma) : \tau$.

Point (1) is obvious, since $f$ can only transform variables to variables. For point (2), note first that $X \in \text{dom}(f(\sigma))$ implies $X \in \text{dom}(\sigma)$ and $f(X) = X$. From $\Gamma_{f(P)} \vdash X : \tau$ we derive that $(X, \tau) \in \Gamma_{f(P)}$. It follows that $X \in \text{dom}(\delta_{f(P)})$ and $\vdash X\delta_{f(P)} : \tau$. Since $\delta_{f(P)} = f(\delta_P)$ by Proposition 3, we get $\vdash Xf(\delta_P) : \tau$.

Since $X \in \text{dom}(\delta_P)$, this is equivalent to $\vdash Xf(\delta_P) : \tau'$ and by Proposition 1, there is a unique ground type $\tau'$ such that $\vdash X\delta_P : \tau'$. Hence, $\Gamma_P \vdash X : \tau'$ by the definition of $\Gamma_P$. Since $\sigma$ is well-typed with respect to $\Gamma_P$, we have $\Gamma_P \vdash X\sigma : \tau'$. 49
From the fact that $X\delta_P$ and $X\sigma$ are both ground and have the same type $\tau'$, we conclude that $f(X\delta_P)$ and $f(X\sigma)$ also have the same type, namely $\tau$. Hence, $\Gamma_{f(P)} \vdash f(X) : \tau$. Since $X \in \text{dom}(f(\sigma))$, we derive $\Gamma_{f(P)} \vdash Xf(\sigma) : \tau$ as required. $\square$

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

**Proof.** First note that $f(\sigma)$ is a well-typed ground substitution by Lemma 29. We show that $(f(tr), f(th), f(\sigma))$ 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 $R \in \mathcal{R}_P$ such that $th(i) = (R, P(R))$. Hence we have

$$f(th)(i) = (R, f(P(R))) = (R, f(P)(R)) \tag{75}$$

Since $(\epsilon, th, \sigma)$ is reachable in $P$, for all $v \in \text{Role}$ and $i \in \text{TID}$ we have $\text{inst}_i(v)\sigma \in \mathcal{A}$. Moreover, we have $\text{inst}_i(v)f(\sigma) = \text{inst}_i(v)\sigma$, we also have

$$\text{inst}_i(v)f(\sigma) \in \mathcal{A} \tag{76}$$

By (75), (76) and $f(\epsilon) = \epsilon$, it is obvious that $(f(\epsilon), f(th), f(\sigma))$ 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

$$(f(tr'), f(th'), f(\sigma)) \text{ is reachable in } f(P) \tag{77}$$

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

* If $r = \text{SEND}$ then there exists $i \in \text{TID}$ and $R \in \mathcal{R}_P$ such that

$$th'(i) = (R, \text{snd}(pt).tl)$$
$$tr = tr'.(i, \text{snd}(pt))$$
$$th = th'[i \mapsto (R, tl)] \tag{78}$$

* If $f(pt) = \text{nil}$ then by (78) we have

$$f(tr) = f(tr')$$
$$f(th) = f(th')[i \mapsto (R, f(tl))] \tag{79}$$

By (79) and (78) we have $f(th) = f(th')$. Together with (77) we obtain that

$$(f(tr), f(th), f(\sigma)) \text{ is reachable in } f(P)$$

50
* If \( f(pt) \neq \text{nil} \) then by (78) we have
\[
\begin{align*}
    f(tr) &= f(tr').(i, \text{snd}(f(pt))) \\
    f(th) &= f(th')[i \mapsto (R, f(tl))] 
\end{align*}
\] (80)
By (78) we have
\[
    f(th')(i) = (R, \text{snd}(f(pt)).f(tl)) 
\] (81)
By (81), (78), (80) and rule \( \text{SEND} \), we have
\[
    (f(tr'), f(th'), f(\sigma)) \rightarrow (f(tr), f(th), f(\sigma)) 
\]
Together with (77) this implies that \( (f(tr), f(th), f(\sigma)) \) is reachable in \( P \).

• If \( r = \text{RECV} \) then there exists \( i \in TID \) and \( R \in \mathcal{R}_P \) such that
\[
    \begin{align*}
        th'(i) &= (R, \text{rcv}(pt).tl) \\
        IK(tr')\sigma \cup IK_0 \vdash pt\sigma 
    \end{align*}
\] (82)
and
\[
    \begin{align*}
        tr &= tr'.(i, \text{rcv}(pt)) \\
        th &= th'[i \mapsto (R, tl)] 
    \end{align*}
\] (83)
If \( f(pt) = \text{nil} \) then similar as before, we have \( (f(tr), f(th), f(\sigma)) \) is reachable in \( f(P) \). Suppose that \( f(pt) \neq \text{nil} \). By (82) and (83) we have
\[
\begin{align*}
    f(tr) &= f(tr').(i, \text{rcv}(f(pt))) \\
    f(th) &= f(th')[i \mapsto (R, f(tl))] 
\end{align*}
\] (84)
To justify the transition \( (f(tr'), f(th'), f(\sigma)) \rightarrow (f(tr), f(th), f(\sigma)) \), it is sufficient to establish the following two premises of rule \( \text{RECV} \):  
1. \( f(th')(i) = (R, \text{rcv}(f(pt)).f(tl)) \), which follows from (82), and 
2. \( IK(f(tr'))f(\sigma) \cup IK_0 \vdash f(pt)f(\sigma) \). This follows from (82) by Proposition 4 since \( \sigma \) is well-typed and ground and from the fact that \( f(IK(tr')) \subseteq IK(f(tr')) \cup \{\text{nil}\} \).
Together with (77) this implies that \( (f(tr), f(th), f(\sigma)) \) is reachable in \( f(P) \).

This completes the proof of the theorem. \( \square \)
D Proof of attack preservation (Section 4)

Theorem (Attack preservation; Justification of Theorem 4). Let \( P \) be a simple-keyed, splitting protocol, \( S_f \) a type-based message transformation for \( P \) and function symbol \( f \), and \( \phi \in L_P \) a \((P,f)\)-safe property. Assume that \( IK_0 \) is simple-keyed and \( f(IK_0) \subseteq IK'_{f(P)} \). Then, for all well-typed states \((tr,th,\sigma)\) reachable in \( P \), we have \((f(tr),f(th),f(\sigma))\) is reachable in \( f(P) \) and \( f(\sigma) \) is well-typed with respect to \( \Gamma_{f(P)} \), and if \((tr,th,\sigma) \not\models \phi \) then \((f(tr),f(th),f(\sigma)) \not\models f(\phi) \).

Proof. Let \((tr,th,\sigma)\) be a reachable state of \( P \) such that \( \sigma \) is well-typed. The first conjunct, namely that \((f(tr),f(th),f(\sigma))\) is reachable in \( f(P) \) and \( f(\sigma) \) is well-typed with respect to \( \Gamma_{f(P)} \), follows from Theorem 3.

To establish the second conjunct, we prove the following generalized statement by induction on the structure of \( \phi \) (which may now contain free thread-id variables).

\[
\forall \theta. (tr,th,\sigma,\theta) \not\models \phi \Rightarrow (f(tr),f(th),f(\sigma),\theta) \not\models f(\phi)
\]

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 \( secret(m) \)) 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 \((tr,th,\sigma,\theta) \not\models \phi \) is equivalent to \((tr,th,\sigma,\theta) \models \neg \phi \) for all atoms \( \phi \) (but not for all formulas, since \( L_P \) is not closed under negation).

- \( \phi \equiv i = j \) or \( \phi \equiv \neg(i = j) \).

\[
(tr,th,\sigma,\theta) \models i = j \\
\Leftrightarrow \theta(i) = \theta(j) \\
\Leftrightarrow (f(tr),f(th),f(\sigma),\theta) \models f(i = j)
\]

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

\[
(tr,th,\sigma,\theta) \models m = m' \\
\Leftrightarrow m\sigma = m'\sigma \\
\Leftrightarrow f(m\sigma) = f(m'\sigma) \quad \text{since} \ \phi \text{ is } (P,f)\text{-safe} \\
\Leftrightarrow f(m)f(\sigma) = f(m')f(\sigma) \quad \text{by Theorem 2} \\
\Leftrightarrow (f(tr),f(th),f(\sigma),\theta) \models f(m) = f(m')
\]

Note that the assumption that \( \phi \) is \((P,f)\)-safe is only needed for the right-to-left direction, i.e., the positive case where \( \phi \equiv (m = m') \).

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

\[
(tr,th,\sigma,\theta) \models \text{role}(i,R) \\
\Leftrightarrow \exists \text{seq} \in \text{Event}^*. \text{th}(\theta(i)) = (R,\text{seq}) \\
\Leftrightarrow \exists \text{seq} \in \text{Event}^*. f(th)(\theta(i)) = (R,f(\text{seq})) \quad f \text{ is the identity on } \mathcal{R}_P^+ \\
\Leftrightarrow (f(tr),f(th),f(\sigma),\theta) \models \text{role}(i,f(R))
\]
Here are the inductive cases:

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

\[
(\text{tr}, \text{th}, \sigma, \vartheta) \models \text{honest}(i, R)
\]
\[
\iff R_{\theta(i)} \sigma \in \mathcal{A}_H
\]
\[
\iff f(R_{\theta(i)} \sigma) \in \mathcal{A}_H \\
\text{f is the identity on } \mathcal{A}
\]
\[
\iff f(R_{\theta(i)} \sigma) \in \mathcal{A}_H \quad \text{by Theorem 2}
\]
\[
\iff (f(\text{tr}), f(\text{th}), f(\sigma), \vartheta) \models \text{honest}(i, f(R))
\]

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

\[
(\text{tr}, \text{th}, \sigma, \vartheta) \models \text{steps}(i, s(m))
\]
\[
\iff (\vartheta(i), s(m)) \in \text{tr}
\]
\[
\iff (\vartheta(i), s(f(m))) \in f(\text{tr}) \\
\text{justified below}
\]
\[
\iff (f(\text{tr}), f(\text{th}), f(\sigma), \vartheta) \models \text{steps}(i, s(f(m)))
\]

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 \((\vartheta(i), s(f(m))) \in f(\text{tr}) \). Then there exists \( s(m') \in Event^2_P \) such that \((\vartheta(i), s(m')) \in \text{tr} \) and \( f(m') = f(m) \). Since \( \phi \) is \((P, f)\)-safe, this implies \( m = m' \) and hence \((\vartheta(i), s(m)) \in \text{tr} \).

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

\[
(\text{tr}, \text{th}, \sigma, \vartheta) \models (i, s(m)) \prec (j, s'(m'))
\]
\[
\iff (\vartheta(i), s(m)) \prec_{\text{tr}} (\vartheta(j), s'(m'))
\]
\[
\iff (\vartheta(i), s(f(m))) \prec_{f(\text{tr})} (\vartheta(j), s'(f(m'))) \\
\text{justified below}
\]
\[
\iff (f(\text{tr}), f(\text{th}), f(\sigma), \vartheta) \models (i, s(f(m))) \prec (j, s'(f(m')))
\]

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 \( \text{nil} \).

For the only-if direction (covering the case that \( \phi \equiv (i, s(m)) \prec (j, s'(m')) \)), suppose \((\vartheta(i), s(f(m))) \prec_{f(\text{tr})} (\vartheta(j), s'(f(m'))). Since \( f \) is order-preserving for events that do not map to \( \text{nil} \), there are events \( s(u), s'(u') \in Event^2_P \) such that \((\vartheta(i), s(u)) \prec_{\text{tr}} (\vartheta(j), s'(u')) \) with \( f(u) = f(m) \) and \( f(u') = f(m') \).

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

- \( \phi \equiv \text{secret}(m) \).

\[
(\text{tr}, \text{th}, \sigma, \vartheta) \neq \text{secret}(m)
\]
\[
\iff IK(\text{tr}) \sigma \cup IK'_0 \vdash m \sigma
\]
\[
\implies f(IK(\text{tr})) f(\sigma) \cup IK'_0 \vdash f(m) f(\sigma) \quad \text{by Proposition 4}
\]
\[
\implies IK(f(\text{tr})) f(\sigma) \cup IK'_0 \vdash f(m) f(\sigma) \quad \text{since } f(\text{IK}(\text{tr})) \subseteq IK(f(\text{tr})) \cup \{\text{nil}\}
\]
\[
\iff (f(\text{tr}), f(\text{th}), f(\sigma), \vartheta) \neq \text{secret}(f(m))
\]

Here are the inductive cases: 53
\[ \phi = \phi_1 \land \phi_2. \]

\[
(tr, th, \sigma, \emptyset) \not\models \phi
\]
\[
\iff (tr, th, \sigma, \emptyset) \not\models \phi_i
\]
\[
\text{for some } i \in \{1, 2\}
\]
\[
\Rightarrow (f(tr), f(th), f(\sigma), \emptyset) \not\models f(\phi_i)
\]
\[
\iff (f(tr'), f(th), f(\sigma), \emptyset) \not\models f(\phi)
\]

- \( \phi = \phi_1 \lor \phi_2. \) Similar to case for conjunction.
- \( \phi = \forall i. \phi'. \)

\[
(tr, th, \sigma, \emptyset) \not\models \forall i. \phi'
\]
\[
\iff (tr, th, \sigma, \emptyset[i \mapsto \text{tid}]) \not\models \phi'
\]
\[
\text{for some tid} \in \text{dom}(th)
\]
\[
\Rightarrow (f(tr), f(th), f(\sigma), \emptyset[i \mapsto \text{tid}]) \not\models f(\phi')
\]
\[
\iff (f(tr'), f(th), f(\sigma), \emptyset) \not\models f(\forall i. \phi')
\]

- \( \phi = \exists i. \phi'. \) Similar to case for universal quantifier.

This concludes the proof of the theorem.

\( \square \)
E Structural transformations

In this section, we present two structural transformations, which directly transform protocol roles. The splitting transformation allows us to split the exchange of pair terms into two exchanges, one for each component of the pair. The relaying transformation removes pure forwarding behavior from a role, i.e., a rcv(X) event followed by a snd(X) event where the variable X does not occur anywhere else in the role.

In the example in the introduction, we have used these transformations in combination to remove the forwarding of tickets by the initiator to the responder in the core Kerberos V protocol (from K5 to K3). More precisely, we have applied two splitting transformations to isolate the responder’s ticket, followed by a relaying transformation to remove the forwarding by the initiator.

We discuss splitting transformations in Section E.1 and relaying transformations in Section E.2. In each case, we first give the definition of the transformation and then we prove its essential properties, namely, well-definedness, simulation, and attack preservation.

E.1 Splitting transformation

We now formally define splitting transformations.

**Definition 25 (Splitting transformations).** Let \( E \subseteq \text{Event} \) be a set of events and a function \( f : \text{Event} \to \text{Event}^\ast \). Then \( f \) is called a splitting transformation with respect to \( E \) iff for all \( s(m) \in \text{Event} \setminus E \), \( f(s(m)) = s(m) \) and for all \( s(m) \in E \),
- \( f(s(m)) = s(m) \) if \( m \) is not a pair,
- \( f(s(m)) = s(m_1) \cdot s(m_2) \) if \( m = \langle m_1, m_2 \rangle \).

We lift \( f \) to threads and traces in the normal way.

**Definition 26 (Splitting transformation on threads and traces).** Let \( P \) be a protocol, \( E \subseteq \text{Event} \) be a set of events and \( f : \text{Event} \to \text{Event}^\ast \) be a splitting transformation with respect to \( E \). We lift \( f \) to a thread \( th \) and a trace \( tr \) of \( P \) as follows:
- \( \forall i \in \text{dom}(th). \ f(th)(i) = (R, f(seq)), \) where \( th(i) = (R, seq) \),
- \( \text{if } tr = \epsilon \text{ then } f(tr) = \epsilon \),
- \( \text{if } tr = (i, s(t)) \cdot tl \text{ then } f(tr) = (i, s(t_1)) \cdots (i, s(t_n)) \cdot f(tl), \) where
  \[ f(s(t)) = s(t_1) \cdots s(t_n), \text{ where } n \geq 1. \]

For a transformed protocol to be executable, we need to transform matching send and receive events in the same way. That is, if we split a send event \( e \), we also split its matching receive event \( e' \) and vice versa. We therefore define a set of splitting events for a protocol \( P \) as follows.

**Definition 27 (Splitting set of events).** Let \( P \) be a protocol. A set of events \( E \subseteq \text{Event}_P \) is a splitting set of \( P \) if, for all matching send and receive events \( \text{snd}(m) \) and \( \text{rcv}(m') \), \( \text{snd}(m) \in E \) if and only if \( \text{rcv}(m') \in E \).
Well-definedness We now show that if \( P \) is a protocol then so is \( f(P) \).

**Proposition 6 (Well-definedness for splitting).** Let \( P \) be a splitting protocol, \( E \) be a splitting set of \( P \) and \( f \) be a splitting transformation on \( \text{Event} \rightarrow \text{Event}^* \) with respect to \( E \cup E^2 \). Then \( f(P) \) is a protocol.

**Proof.** We need to show three points for \( f(P) \):

1. the sets of variables and fresh values occurring in different roles are pairwise disjoint,
2. for all \( e \in f(P)(R) \) and \( X \in \text{vars(term}(e)) \), there is an event \( \text{rcv}(t) \) in \( f(P)(R) \) such that \( \text{rcv}(t) \) equals or precedes \( e \) in \( P(R) \) and \( X \in \text{acc}(t) \),
3. there exists the honest substitution \( \delta_{f(P)} \).

The first point follows directly from Definition 25. We now show the second point. Let \( e \in f(P)(R) \) and \( X \in \text{vars(term}(e)) \). Then since \( P \) is non-overlapping, there exists a unique event \( e' \in P(R) \) such that

\[
\begin{align*}
&f(e') = e_1 \cdots e_n, \text{ where } n \geq 1, \\
e \in \{ e_1, \ldots, e_n \}, \\
\text{term}(e') = \langle \text{term}(e_1), \ldots, \text{term}(e_n) \rangle. 
\end{align*}
\]

Since \( X \in \text{vars(term}(e')) \), by (85), we have \( X \in \text{vars(term}(e')) \). This, since \( P \) is a protocol, implies that there is an event \( \text{rcv}(t) \in \text{set}(P(R)) \) such that \( \text{rcv}(t) \) equals or precedes \( e' \) in \( P(R) \) and \( X \in \text{acc}(t) \). Moreover, by Definition 25 we have

\[
\begin{align*}
&f(\text{rcv}(t)) = \text{rcv}(t_1) \cdots \text{rcv}(t_m), \text{ where } m \geq 1, \\
&t = \langle t_1, \ldots, t_m \rangle. 
\end{align*}
\]

Since \( X \in \text{acc}(t) \), by (86), there is \( j \in \{ 1, \ldots, m \} \) such that \( X \in \text{acc}(t_j) \). Since \( \text{rcv}(t) \) equals or precedes \( e' \) in \( P(R) \), \( \text{rcv}(t_j) \) equals or precedes \( e \) in \( f(P)(R) \). Moreover, we have \( \text{rcv}(t_j) \in \text{set}(f(P)(R)) \). Hence the second point holds.

It remains to show that there exists a honest substitution \( \delta_{f(P)} \). Let \( S_P = \{ (s_1, r_1), \ldots, (s_m, r_m) \} \) be the list of matching send and receive events of \( P \). Since \( P \) is a protocol, there exists the honest substitution \( \delta_P \) such that:

\[
\begin{align*}
&- \delta_1 = \text{mgu}(\text{term}(s_1), \text{term}(r_1)), \\
&- \delta_k = \text{mgu}(\text{term}(s_k)(\delta_{k-1} \circ \cdots \circ \delta_1), \text{term}(r_k)(\delta_{k-1} \circ \cdots \circ \delta_1)) \text{ for } 1 < k \leq m, \\
&- \delta_P = \delta_m \circ \cdots \circ \delta_1. 
\end{align*}
\]

Since \( P \) is a protocol, for all \( k \in \{ 1, \ldots, m \} \) and all variables \( x \in \text{vars}(s_k) \) there exists a previous receive event \( r_{k_i} \) such that \( x \in \text{vars}(r_{k_i}) \). That means for all \( k \in \{ 1, \ldots, m \} \) we have \( \text{term}(s_k)(\delta_{k-1} \circ \cdots \circ \delta_1) \) is ground. Hence we have

\[
\begin{align*}
&\text{term}(s_k) = \text{term}(r_k) \delta_1, \\
&\text{term}(s_k)(\delta_{k-1} \circ \cdots \circ \delta_1) = (\text{term}(r_k)(\delta_{k-1} \circ \cdots \circ \delta_1)) \delta_k \text{ for } 1 < k \leq m. 
\end{align*}
\]

By Definition 25, for all \( k \in \{ 1, \ldots, m \} \) we have

\[
\begin{align*}
&f(\text{snd}(s_k)) = \text{snd}(s_{i_{k,1}}) \cdots \text{snd}(s_{i_{k,n_k}}) \text{ for } n_k \geq 1 \\
&s_k = \langle s_{i_{k,1}}, \ldots, s_{i_{k,n_k}} \rangle \\
&f(\text{rcv}(r_k)) = \text{rcv}(r_{i_{k,1}}) \cdots \text{rcv}(r_{i_{k,m_k}}) \text{ for } m_k \geq 1 \\
&r_k = \langle r_{i_{k,1}}, \ldots, r_{i_{k,m_k}} \rangle. 
\end{align*}
\]
Since $P$ is splitting, for all $k \in \{1, \ldots, m\}$ we have $n_k = m_k$. Hence the list of matching send and receive events of $f(P)$ is given by

$$S_{f(P)} = [(s_{i_1}, r_{i_1}), \ldots, (s_{i_n}, r_{i_n})].$$

This by (87) and (88) yields that for all $k \in \{1, \ldots, m\}$ and all $j \in \{i_k, 1, \ldots, i_k, n_k\}$, $\delta_k = mgu(s_j, r_j)$. Hence $\delta_P$ is also the honest substitution of $f(P)$, i.e., $\delta_{f(P)} = \delta_P$. This completes the proof of the proposition. □

**Simulation** We show a similar simulation result for splitting transformations in the following proposition.

**Proposition 7 (Simulation for splitting).** Let $P$ be a splitting protocol, $E$ be a splitting set of $P$ and $f : \text{Event} \to \text{Event}^*$ be a splitting transformation with respect to $E \cup E^\circ$. Assume that $IK_0 \subseteq IK'_0$. Then for all states $(tr, th, \sigma)$ reachable in $P$, we have $(f(tr), f(th), \sigma)$ is reachable in $f(P)$.

**Proof.** We prove the proposition by induction on the number $n$ of transitions leading to a state $(tr, th, \sigma)$.

- **Base case** ($n = 0$): We have $tr = \epsilon$. Moreover, for all $i \in TID$, there exists $R \in \mathcal{R}_P$ such that

$$th(i) = (R, \text{inst}_i(P(R)))$$

$$f(th)(i) = (R, \text{inst}_i(f(P(R))))$$

Since $f(\epsilon) = \epsilon$, (89) implies that $(f(tr), f(th), \sigma) \in \text{Init}(P)$ and hence $(f(tr), f(th), \sigma)$ is reachable in $f(P)$.

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

$$(f(tr'), f(th'), \sigma) \text{ is reachable in } f(P)$$

(90)

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 $R \in \mathcal{R}_P$ such that

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

$$tr = tr' \cdot (i, \text{snd}(pt))$$

$$th = th'[i \mapsto (R, tl)]$$

(91)

Moreover, by (91) we have

$$f(th) = f(th'[i \mapsto (R, tl)]) = f(th'[i \mapsto (R, f(tl))]),$$

$$f(tr) = f(tr' \cdot (i, \text{snd}(pt))) = f(tr' \cdot (i, \text{snd}(t_1)) \cdots (i, \text{snd}(t_n))),$$

where $n \geq 1$.

(92)
For all $j \in \{1,\ldots,n\}$, let
\[ tr_j = f(tr') \cdot (i, \text{snd}(t_1)) \cdots (i, \text{snd}(t_j)), \]
\[ th_j = f(th')[i \mapsto (R, \text{snd}(t_j) \cdots \text{snd}(t_n) \cdot f(tl))]. \]

(93)

Then by (92), for all $j \in \{1,\ldots,n\}$ we have
\[ th_j(i) = (R, \text{snd}(t_j) \cdots \text{snd}(t_n) \cdot f(tl)) \]

(94)

By (92), (93), (94) and rule \textit{SEND}, there is a sequence of transitions

\[ (f(tr'), f(th'), \sigma) \rightarrow (tr_1, th_1, \sigma) \rightarrow (tr_2, th_2, \sigma) \rightarrow \cdots \rightarrow (tr_n, th_n, \sigma) \]

where $(tr_n, th_n, \sigma) = (f(tr), f(th), \sigma)$

This together with (90) imply that $(f(tr), f(th), \sigma)$ is reachable in $P$.

- If $r = \text{RECV}$ then there exists $i \in TID$ and $R \in \mathcal{R}_P$ such that

\[ th'_i(i) = (R, \text{rcv}(pt) \cdot tl) \]
\[ tr = tr' \cdot (i, \text{rcv}(pt)) \]
\[ th = th'_i[i \mapsto (R, tl)] \]

(95)

By (95) we have
\[ f(th')(i) = f(th'_i(i)) = (R, \text{rcv}(t_1) \cdots \text{rcv}(t_n) \cdot f(tl)), \] where $n \geq 1$

(96)

Moreover, it follows from Definition 25 that
\[ \text{set}(\text{split}(IK(tr'))) = \text{set}(\text{split}(IK(f(tr')))) \]

This by Lemma 18 yields
\[ \text{set}(\text{split}(IK(tr')\sigma)) = \text{set}(\text{split}(IK(f(tr'))\sigma)) \]

Hence since $IK_0 \subseteq IK'_0$, by (95) we have $IK(f(tr')\sigma) \cup IK'_0 \vdash pt\sigma$. By Definition 25 we have
\[ \text{set}(\text{split}([t_1,\ldots,t_n])) = \text{set}(\text{split}(pt)) \]

Therefore, for all $i \in \{1,\ldots,n\}$ we have
\[ IK(f(tr')\sigma) \cup IK'_0 \vdash t_i\sigma \]

(97)

For all $j \in \{1,\ldots,n\}$, let
\[ tr_j = f(tr') \cdot (i, \text{rcv}(t_1)) \cdots (i, \text{rcv}(t_j)), \]
\[ th_j = f(th'[i \mapsto (R, \text{rcv}(t_j) \cdots \text{rcv}(t_n) \cdot f(tl))]). \]

(98)

Then by (98), for all $j \in \{1,\ldots,n\}$ we have
\[ th_j(i) = \text{rcv}(t_j) \cdots \text{rcv}(t_n) \cdot f(tl) \]

(99)
By (97), (98), (99), (96) and rule $RECV$, there is a sequence of transitions

$$(f(tr'), f(th'), \sigma) \rightarrow (tr_1, th_1, \sigma) \rightarrow (tr_2, th_2, \sigma) \rightarrow \cdots \rightarrow (tr_n, th_n, \sigma)$$

where $((tr_n, th_n, \sigma)) = (f(tr), f(th), \sigma)$

Hence by (90) we have $(f(tr), f(th), \sigma)$ is reachable in $P$.

This completes the proof of the proposition.

Next, we show that splitting transformations preserve attacks.

**Attack preservation** We first lift splitting transformations to formulas as follows.

**Definition 28 (Splitting transformations on formulas).** Let $\phi \in L_P$ and $f$ be a splitting transformation on $P$. We define $f(\phi)$ as follows:

- $f(i = i') = i = i'$
- $f(m = m') = m = m'$
- $f(\text{role}(i, R)) = \text{role}(i, R)$
- $f(\text{honest}(i, R)) = \text{honest}(i, R)$
- $f(\text{steps}(i, e)) = \text{steps}(i, e_1) \wedge \ldots \wedge \text{steps}(i, e_n)$
- $f((i, e) < (j, e')) = (i, e_n) < (j, e'_1)$
- $f(\text{secret}(m)) = \text{secret}(m)$
- $f(\neg A) = \neg f(A)$
- $f(\phi_1 \wedge \phi_2) = f(\phi_1) \wedge f(\phi_2)$
- $f(\phi_1 \vee \phi_2) = f(\phi_1) \vee f(\phi_2)$
- $f(\forall_i. \phi') = \forall_i. f(\phi')$
- $f(\exists_i. \phi') = \exists_i. f(\phi')$

where

- $f(e) = e_1 \cdots e_n$,
- $f(e') = e'_1 \cdots e'_m$, and
- $n, m \geq 1$

For attack preservation to hold, we require that every event in a transformed protocol is split from a unique event in the original protocol. Intuitively, this condition is necessary for properties involving occurrences of events and ordering of events. We now formally define a class of protocols that satisfy the condition.

**Definition 29.** A protocol $P$ is non-overlapping if $\text{set}(\text{split}(t)) \cap \text{set}(\text{split}(t')) = \emptyset$ holds for all $\{s(t), s(t')\} \subseteq \text{Event}_P$.

We show attack preservation for non-overlapping protocols in the following theorem.

**Theorem 5 (Attack preservation for splitting).** Let $E \subseteq \text{Event}$ be a set of events, $P$ be a non-overlapping and splitting protocol and $f: \text{Event}_P \cup \text{Event}_P^* \rightarrow \text{Event}^*$ be a splitting transformation with respect to $E$ and $\phi \in L_P$ be closed. Then we have, for all states $(tr, th, \sigma)$ reachable in $P$,
- \((f(tr), f(th), f(\sigma))\) is reachable in \(f(P)\) and
- \((tr, th, \sigma) \not\equiv \phi\) implies \((f(tr), f(th), f(\sigma)) \not\equiv f(\phi)\).

Proof. Let \((tr, th, \sigma)\) be a reachable state of \(P\). The first conjunct, namely that \((f(tr), f(th), \sigma)\) is reachable in \(f(P)\) follows from Proposition 7. To show the second conjunct, we prove the following statement by induction on the structure of \(\phi\):
\[
\forall \vartheta. (tr, th, \sigma, \vartheta) \not\equiv \phi \Rightarrow (f(tr), f(th), \sigma, \vartheta) \not\equiv f(\phi)
\]

We consider the following cases:

- \(\phi \equiv i = j\) or \(\phi \equiv \neg(i = j)\).
  
  \[
  \begin{align*}
  (tr, th, \sigma, \vartheta) &\models i = j \\
  \iff & \vartheta(i) = \vartheta(j) \\
  \iff & (f(tr), f(th), \sigma, \vartheta) \models f(i = j)
  \end{align*}
  \]

- \(\phi \equiv m = m'\) or \(\phi \equiv \neg(m = m')\).
  
  \[
  \begin{align*}
  (tr, th, \sigma, \vartheta) &\models m = m' \\
  \iff & m\sigma = m'\sigma \\
  \iff & (f(tr), f(th), \sigma, \vartheta) \models m = m'
  \end{align*}
  \]

- \(\phi \equiv \text{role}(i, R)\) or \(\phi \equiv \neg\text{role}(i, R)\).
  
  \[
  \begin{align*}
  (tr, th, \sigma, \vartheta) &\models \text{role}(i, R) \\
  \iff & \exists \text{seq} \in \text{Event}^*. th(\vartheta(i)) = (R, \text{seq}) \\
  \iff & \exists \text{seq} \in \text{Event}^*. f(th)(\vartheta(i)) = (R, f(\text{seq})) \text{ by Definition 26} \\
  \iff & (f(tr), f(th), \sigma, \vartheta) \models \text{role}(i, R)
  \end{align*}
  \]

- \(\phi \equiv \text{honest}(i, R)\) or \(\phi \equiv \neg\text{honest}(i, R)\).
  
  \[
  \begin{align*}
  (tr, th, \sigma, \vartheta) &\models \text{honest}(i, R) \\
  \iff & R_{\vartheta(i)}\sigma \in A_H \\
  \iff & (f(tr), f(th), \sigma, \vartheta) \models \text{honest}(i, R)
  \end{align*}
  \]

- \(\phi \equiv \text{steps}(i, s(m))\) or \(\phi \equiv \neg\text{steps}(i, s(m))\), where \(s \in \{\text{snd, rcv}\}\).
  
  \[
  \begin{align*}
  (tr, th, \sigma, \vartheta) &\models \text{steps}(i, s(m)) \\
  \iff & (\vartheta(i), s(m)) \in tr
  \end{align*}
  \]

We now show that \((\vartheta(i), s(m)) \in tr \iff (\vartheta(i), f(s(m))) \in f(tr)\). The direction \((\vartheta(i), s(m)) \in tr \Rightarrow (\vartheta(i), f(s(m))) \in f(tr)\) is obvious by Definition 26. It remains to show that \((\vartheta(i), f(s(m))) \in f(tr) \Rightarrow (\vartheta(i), s(m)) \in tr\). By Definition 25 we have

\[
f(s(m)) = s(m_1) \cdots s(m_n), \text{ where } n \geq 1
\]

\[
\text{set(split}(m)) = \text{set}(\text{split}(\{m_1, \ldots, m_n\}))
\]

\[\tag{100}\]
Then by Definition 26, \((\vartheta(i), f(s(m))) \in f(tr)\) implies that there exists \(s(m') \in \text{Event}^t_P\), such that

\[
(\vartheta(i), s(m')) \in tr \\
f(s(m')) = s(m'_1) \cdots s(m'_k), \text{ where } k \geq 1, \\
set(\text{split}(m')) = \set(\text{split}(\{m'_1, \ldots, m'_k\})) \\
m_1 \in \{m'_1, \ldots, m'_k\}
\]

(101)

By (100) and (101) we have

\[
\text{set}(\text{split}(m_1)) \subseteq \text{set}(\text{split}(m)) \cap \text{set}(\text{split}(m'))
\]

(102)

If \(m \neq m'\) then by (102) and Definition 29 we conclude that \(P\) is not a non-overlapping protocol which is a contradiction. Hence \(m' = m\) which by (101) implies \((\vartheta(i), s(m)) \in tr\). Therefore, we have

\[
(\vartheta(i), s(m)) \iff (\vartheta(i), f(s(m))) \in f(tr)
\]

Moreover, we also have

\[
(\vartheta(i), f(s((m)))) \in f(tr) \iff (f(tr), f(th), \sigma, \vartheta) \models f(\text{steps}(i, s(m)))
\]

Hence it holds that

\[
(tr, th, \sigma, \vartheta) \models \text{steps}(i, s(m)) \iff (f(tr), f(th), \sigma, \vartheta) \models f(\text{steps}(i, s(m)))
\]

\[- \phi \equiv (i, s(m)) \prec (j, s'(m')) \text{ or } \phi \equiv \neg((i, s(m)) \prec (j, s'(m'))),
\]

where \(s, s' \in \{\text{snd, rcv}\}.

\[
(tr, th, \sigma, \vartheta) \models (i, s(m)) \prec (j, s'(m')) \iff (\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m'))
\]

We now show that

\[
(\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m')) \iff (\vartheta(i), f(s(m))) \prec_{f(tr)} (\vartheta(j), f(s'(m')))
\]

By Definition 25 we have

\[
f(s(m)) = s(m_1) \cdots s(m_n) \\
f(s'(m')) = s'(m'_1) \cdots s'(m'_k) \\
n, m \geq 1
\]

By Definition 26, it is obvious that

\[
(\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m')) \Rightarrow (\vartheta(i), s(m)) \prec_{f(tr)} (\vartheta(j), s'(m'_1))
\]

We need to show that

\[
(\vartheta(i), s(m)) \prec_{f(tr)} (\vartheta(j), s'(m'_1)) \Rightarrow (\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m'))
\]

(103)
Suppose \( (\vartheta(i), s(m_n)) \prec_{f(tr)} (\vartheta(j), s'(m'_1)) \). Then there exists \( s(t), s'(t') \in tr \) such that
\[
\begin{align*}
(\vartheta(i), s(t)) & \prec_{tr} (\vartheta(j), s'(t')) \\
f(s(t)) &= s(t_1) \cdots s(t_1), \text{ where } l \geq 1 \text{ and } m_n \in \{ t_1, \ldots, t_l \} \\
f(s'(t')) &= s'(t'_1) \cdots s'(t'_v), \text{ where } v \geq 1 \text{ and } m'_1 \in \{ t'_1, \ldots, t'_v \}
\end{align*}
\] (104)

Similar as above, since \( P \) is non-overlapping we must have \( t = m \) and \( t' = m' \). This by (104) implies \( (\vartheta(i), s(m)) \prec_{tr} (\vartheta(j), s'(m')) \). Hence (103) holds. Moreover, we have
\[
\begin{align*}
(f(tr), f(th), \sigma, \vartheta) &= f((i, s(m)) \prec (j, s'(m'))) \\
\Leftrightarrow (\vartheta(i), s(m_n)) & \prec_{f(tr)} (\vartheta(j), s'(m'_1))
\end{align*}
\]

Therefore, we have
\[
\begin{align*}
(tr, th, \sigma, \vartheta) & \models (i, s(m)) \prec (j, s'(m')) \\
\Leftrightarrow (f(tr), f(th), \sigma, \vartheta) & \models f((i, s(m)) \prec (j, s'(m'))))
\end{align*}
\]

- \( \phi \equiv \text{secret}(m) \).
\[
\begin{align*}
(tr, th, \sigma, \vartheta) & \nvdash \text{secret}(m) \\
\Leftrightarrow IK(tr)\sigma \cup IK_0 \vdash m\sigma \\
\Rightarrow f(IK(tr))\sigma \cup IK'_0 \vdash m\sigma \quad \text{since } IK_0 \subseteq IK'_0, \text{ and} \\
\quad \text{split}(f(IK(tr))) = \text{split}(IK(tr)) \\
\Leftrightarrow (f(tr), f(th), \sigma, \vartheta) \nvdash \text{secret}(m)
\end{align*}
\]

Here are the inductive cases:

- \( \phi = \phi_1 \land \phi_2 \).
\[
\begin{align*}
(tr, th, \sigma, \vartheta) & \nvdash \phi \\
\Leftrightarrow (tr, th, \sigma, \vartheta) & \nvdash \phi_i \quad \text{for some } i \in \{ 1, 2 \} \\
\Rightarrow (f(tr), f(th), \sigma, \vartheta) & \nvdash f(\phi_i) \text{ by induction hypothesis} \\
\Leftrightarrow (f(tr), f(th), \sigma, \vartheta) & \nvdash f(\phi)
\end{align*}
\]

- \( \phi = \phi_1 \lor \phi_2 \). Similar to case for conjunction.

- \( \phi = \exists i. \phi' \).
\[
\begin{align*}
(tr, th, \sigma, \vartheta) & \nvdash \exists i. \phi' \\
\Leftrightarrow (tr, th, \sigma, \vartheta[i \mapsto \text{tid}] & \nvdash \phi' \quad \text{for some } \text{tid} \in \text{dom}(th) \\
\Rightarrow (f(tr), f(th), \sigma, \vartheta[i \mapsto \text{tid}] & \nvdash f(\phi') \text{ by induction hypothesis} \\
\Leftrightarrow (f(tr), f(th), \sigma, \vartheta) & \nvdash f(\forall i. \phi')
\end{align*}
\]

- \( \phi = \forall i. \phi' \). Similar to case for universal quantifier.

This completes the proof of the theorem. \( \square \)
E.2 Relaying transformation

In this section, we present relaying transformation which allows a sender to directly send messages to the intended receiver without using a third participant as an intermediate. Intuitively, the third agent receives a message from the sender into a variable (since he cannot verify this message) and forward this message (under this variable) to the sender’s intended receiver without using this variable in any further messages. Hence removing this variable from the third participant’s role description allows the sender to directly send his message to the receiver.

We characterize a set of variables used in such a communication by defining independent sets of protocol variables.

**Definition 30 (Independent sets of variables).** Let $T$ be a set of terms. A set of variables $X$ is called independent with respect to $T$ iff

- $X \subseteq \text{St}(T)$,
- for all $x \in X$ and all $t \in T$, $x \in \text{St}(t)$ implies $x = t$.

We now define relaying transformation as a message transformation fashion.

**Definition 31 (Relaying transformations).** Let $f : T \to T$ be a function on $T$ and $X$ be a set of variables such that $X \subseteq \text{St}(T)$. Then $f$ is called a relaying transformation on $T$ with respect to $X$ iff for all $t \in T$ the following holds:

- if $t \in X$ then $f(t) = \text{nil}$,
- if $t$ is an atom or a variable such that $t \notin X$ then $f(t) = t$,
- if $t = h(u)$ then $f(t) = h(f(u))$,
- if $t = (t_1, t_2)$ then $f(t) = (f(t_1), f(t_2))$,
- if $t = \{u\}_k$ then $f(t) = \{f(u)\}_{f(k)}$.

From this definition, we can show that relaying transformation either removes a term or keeps the term unchanged as in the following corollary.

**Corollary 4.** Let $T \subseteq \mathcal{T}$ be a set of terms, $X$ be an independent set of variables to $T$ and $f$ be a relaying transformation on $T$ with respect to $X$. Then for all $t \in T$, $f(t) = t$ or $f(t) = \text{nil}$. Moreover, $f(t) \neq t \iff t \in X$.

Interestingly, over normal terms, our message transformations cover relaying transformation.

**Proposition 8.** $f$ is a message transformation on $nf(T)$.

To lift relaying transformation to protocol specifications, we define the notion of independent sets of variables with respect to protocols.

**Definition 32 (Independent sets for protocol variables).** Let $P$ be a protocol. A set of variables $X$ is called independent for $P$ iff the following holds:

- $X$ is independent of $Rt_P$,
- for all $x \in X$, there exists $R \in \mathcal{R}_P$ such that $\{\text{rcv}(x), \text{snd}(x)\} \subseteq \text{set}(P(R))$. 

63
Well-definedness We show that if $P$ is a protocol then so is $f(P)$.

**Proposition 9 (Well-definedness for relaying).** Let $P$ be a protocol, $\mathcal{X}$ be the independent set of $P$ and $f$ be a relaying transformation on $\mathcal{M}_P$ with respect to $\mathcal{X} \cup \mathcal{X}^2$. Then $f(P)$ is a protocol.

**Proof.** We need to show three points for $f(P)$:

1. the sets of variables and fresh values occurring in different roles are pairwise disjoint,
2. for all $e \in f(P)(R)$ and $X \in \text{vars}(\text{term}(e))$, there is an event $\text{rcv}(t)$ in $f(P)(R)$ such that $\text{rcv}(t)$ equals or precedes $e$ in $P(R)$ and $X \in \text{acc}(t)$,
3. there exists the honest substitution $\delta_{f(P)}$.

Points 1 follows from Corollary 4 and the fact that $P$ is a protocol. To show Point 2, we assume that $e \in f(P)(R)$ and $X \in \text{vars}(\text{term}(e))$. Then it is obvious that $f(X) = X$ and thus $X \notin \mathcal{X}$. This by Corollary 4 implies that $e \in P(R)$ and $f(e) = e$. Since $X \in \text{vars}(\text{term}(e))$ and $P$ is a protocol, there is an event $\text{rcv}(t) \in \text{set}(P(R))$ such that $\text{rcv}(t)$ equals or precedes $e$ in $P(R)$ and

$$X \in \text{acc}(t)$$

(105)

Since $X \notin \mathcal{X}$, by Corollary 4, we have $f(\text{rcv}(t)) = \text{rcv}(t)$ and thus we also have $\text{rcv}(t) \in \text{set}(f(P)(R))$. This together with (105) yield that point 2 holds.

It remains to show that there exists the honest substitution $\delta_{f(P)}$.

Let $S_P = [(s_1, r_1), \ldots, (s_m, r_m)]$ be the list of matching send and receive events of $P$. It is easy to see that the list of matching send and receive events of $f(P)$ is given by

$$S_{f(P)} = [(s_{i_1}, r_{j_1}), \ldots, (s_{i_n}, r_{j_n})],$$

where $1 \leq n \leq m$, $1 \leq i_1 < \ldots < i_n \leq m$, $1 \leq j_1 < \ldots < j_n \leq m$, for all $k \in \{1, \ldots, m\}$, $f(\text{term}(s_k)) \neq \text{nil} \iff k \in \{i_1, \ldots, i_n\}$

and for all $k \in \{1, \ldots, n\}$,

- if $f(\text{term}(r_k)) \neq \text{nil}$ then $j_k = i_k$,
- if $f(\text{term}(r_k)) = \text{nil}$ then $j_k = v$ such that $\text{term}(s_v) = \text{term}(r_k)$.

Note that for all $k \in \{2, \ldots, n\}$ we have

$$j_{k-1} = i_k - 1$$

(106)

Since $P$ is a protocol, we also have $i_1 = 1$. Moreover, there exists the honest substitution $\delta_P$ such that

- $\delta_1 = \text{mgu}(\text{term}(s_1), \text{term}(r_1))$,
- $\delta_k = \text{mgu}(\text{term}(s_k)(\delta_{k-1} \circ \cdots \circ \delta_1), \text{term}(r_k)(\delta_{k-1} \circ \cdots \circ \delta_1))$ for $1 < k \leq m$,
- $\delta_P = \delta_m \circ \cdots \circ \delta_1$. 
We show that for all \( k \in \{1, \ldots, n\} \), the following holds
\[
\delta_{j_k} \circ \delta_{j_k-1} \circ \cdots \circ \delta_{i_k} = mgu(t_{i_k}(\delta_{i_k-1} \circ \cdots \circ \delta_1), \ u_{j_k}(\delta_{i_k-1} \circ \cdots \circ \delta_1)), \tag{107}
\]
where we abbreviate \( t_v = \text{term}(s_v) \) and \( u_v = \text{term}(r_v) \).

Since \( P \) is a protocol, for all \( k \in \{1, \ldots, m\} \) and all variables \( x \in \text{vars}(s_{i_k}) \) there exists a previous receive event \( r_{i_k} \) such that \( x \in \text{vars}(r_{i_k}) \). That means for all \( k \in \{1, \ldots, m\} \) we have \( t_k(\delta_{k-1} \circ \cdots \circ \delta_1) \) is ground. Hence we have
\[
t_1 = u_1 \delta_1, \quad t_k(\delta_{k-1} \circ \cdots \circ \delta_1) = (u_k(\delta_{k-1} \circ \cdots \circ \delta_1)) \delta_k \text{ for } 1 < k \leq m. \tag{108}
\]

Let \( k \in \{1, \ldots, n\} \). We consider two cases:

- If \( f(u_{j_k}) = \text{nil} \) then by (108) we have
  \[
  t_{i_k}(\delta_{i_k-1} \circ \cdots \circ \delta_1) = (u_{i_k}(\delta_{i_k-1} \circ \cdots \circ \delta_1)) \delta_{i_k},
  \tag{109}
  \]

  Since \( f(u_{i_k}) = \text{nil} \), we have \( u_{i_k} \in \mathcal{X} \). Therefore, \( u_{i_k} \) and thus \( t_{j_k} \) does not occur anywhere else, except in \( r_{i_k} \) and \( s_{j_k} \). Hence by (109) we obtain
  \[
  t_{i_k}(\delta_{i_k-1} \circ \cdots \circ \delta_1) = (u_{j_k}(\delta_{j_k-1} \circ \cdots \circ \delta_1)) \delta_{j_k}. \tag{110}
  \]

  It is obvious that \( j_k > i_k \). Hence by (110) we obtain (107).

- If \( f(u_{i_k}) \neq \text{nil} \) then \( j_k = i_k \). This by (109) yields (107).

Let \( \gamma_{i_k} = \delta_{j_k} \circ \delta_{j_k-1} \circ \cdots \circ \delta_{i_k} \). We now show that for all \( k \in \{2, \ldots, n\} \), the following holds
\[
\gamma_{i_{k-1}} \circ \cdots \circ \gamma_{i_1} = \delta_{i_{k-1}} \circ \cdots \circ \delta_1. \tag{111}
\]

We prove (111) by induction on \( k \).

- Base case \((k = 2)\): It follows directly from (106) that (111) holds.

- Inductive case \((2 < k \leq n)\): We have
  \[
  \gamma_{i_k} \circ \cdots \circ \gamma_{i_1} = \gamma_{i_k} \circ \gamma_{i_{k-1}} \circ \cdots \circ \gamma_{i_1} = \delta_{i_k} \circ \delta_{j_{k-1}} \circ \cdots \circ \delta_{i_k} \circ \gamma_{i_{k-1}} \circ \cdots \circ \gamma_{i_1}. \tag{112}
  \]

  By induction hypothesis we have
  \[
  \gamma_{i_{k-1}} \circ \cdots \circ \gamma_{i_1} = \delta_{i_{k-1}} \circ \cdots \circ \delta_1 \tag{113}
  \]

  Hence by (112) and (113) we have
  \[
  \gamma_{i_k} \circ \cdots \circ \gamma_{i_1} = \delta_{i_k} \circ \delta_{j_{k-1}} \circ \cdots \circ \delta_{i_k} \circ \delta_{i_{k-1}} \circ \cdots \circ \delta_1 = \delta_{i_k} \circ \delta_{j_{k-1}} \circ \cdots \circ \delta_1. \tag{114}
  \]

By (106) we have \( j_k = i_{k+1} - 1 \). Hence by (114) we obtain (111).

65
Finally, we show that for all \( k \in \{1, \ldots, n\} \),

\[
\gamma_{ik} = mgu(t_{ik}(\gamma_{i-1} \circ \cdots \circ \gamma_{i1}), u_{ik}(\gamma_{i-1} \circ \cdots \circ \gamma_{i1})).
\] (115)

We consider two cases:

- If \( k = 1 \) then \( i_k = 1 \). Hence (115) follows directly from (107).
- If \( k > 1 \) then (115) follows from (107) and (111).

Therefore (115) holds. It implies that there exists the honest substitution of \( f(P) \), namely \( \delta_{f(P)} = \gamma_{i_n} \circ \cdots \circ \gamma_{i1} \). This completes the proof of the proposition.

We next show simulation result for relaying transformation.

**Simulation** To show executability, we first show the following deducibility preservation result.

**Lemma 30.** Let \( P \) be a protocol, \( \mathcal{X} \) be the independent set of \( P \) and \( f \) be a relaying transformation on \( \mathcal{M}_P \) with respect to \( \mathcal{X} \cup \mathcal{X}^2 \). Assume that \( IK_0 \subseteq IK'_0 \). Then for all states \((tr, th, \sigma)\) reachable in \( P \) and all \( pt \in \mathcal{M}_p \), we have \( IK(tr)\sigma \cup IK_0 \vdash pt\sigma \) implies \( IK(f(tr))f(\sigma) \cup IK'_0 \vdash f(pt)f(\sigma) \).

**Proof.** We assume that

\[
IK(tr)\sigma \cup IK_0 \vdash pt\sigma
\] (116)

Moreover, we have

\[
IK_0 \subseteq IK'_0
\] (117)

By Corollary 4, we have \( f(pt) = pt \) or \( f(pt) = \text{nil} \).

- If \( f(pt) = \text{nil} \) then since \( \text{nil} \in IK'_0 \), it is obvious that \( IK(f(tr))f(\sigma) \cup IK'_0 \vdash f(pt)f(\sigma) \).
- If \( f(pt) = pt \) then by Corollary 4 we have

\[
\mathcal{X} \cap \text{vars}(pt) = \emptyset
\] (118)

We consider two cases:

- If \( \mathcal{X} \cap \text{vars}(tr) = \emptyset \) then by Corollary 4 we have

\[
f(tr) = tr
\forall x \in \text{dom}(\sigma) \setminus \mathcal{X}. x\sigma = xf(\sigma)
\] (119)

By (116), (117), (118) and (119) we obtain

\[
IK(f(tr))f(\sigma) \cup IK'_0 \vdash f(pt)f(\sigma)
\]
If $X \cap \text{vars}(tr') \neq \emptyset$ then for all $x \in X \cap \text{vars}(tr)$, there must be prefixes $tr_1, tr_2$ of $tr$ and $th_1, th_2$ such that

\[
x \notin \text{vars}(tr_1)
tr_2 = tr_1 \cdot (j, \text{rcv}(x)) \text{ for some } j \in \text{TID}
(tr_1, th_1, \sigma) \text{ is reachable in } P
(tr_1, th_1, \sigma) \rightarrow (tr_2, th_2, \sigma) \rightarrow^*(tr, th, \sigma)
\]  

(120)

By (120), the transition $(tr_1, th_1, \sigma) \rightarrow (tr_2, th_2, \sigma)$ must be obtained by applying rule $\text{RECV}$. Hence we have

\[
\text{IK}(tr_1) \sigma \cup \text{IK}_0 \vdash x \sigma
\]  

(121)

Moreover, since $x \notin \text{vars}(tr_1)$, by Corollary 4 we have $f(tr_1) = tr_1$. Hence $tr_1 \subseteq f(tr)$. This by (116), (117), (119) and (121) yields

\[
\text{IK}(f(tr)) f(\sigma) \cup \text{IK}_0' \vdash x \sigma
\]  

(122)

Moreover, we have

\[
\text{IK}(f(tr)) = \text{IK}(tr) \setminus X
\forall y \in \text{dom}(\sigma) \setminus X. \ y \sigma = y f(\sigma)
\]  

(123)

By (116), (117), (122) and (123) we have

\[
\text{IK}(f(tr)) f(\sigma) \cup \text{IK}_0' \vdash f(pt) f(\sigma)
\]

This completes the proof of the lemma. \qed

We are now ready to show simulation for relaying transformation.

**Proposition 10 (Simulation for relaying).** Let $P$ be a protocol, $X$ be the independent set of $P$ and $f$ be a relaying transformation on $M^P_2$ with respect to $X \cup X'$. Assume that $\text{IK}_0 \subseteq \text{IK}_0'$. Then for all states $(tr, th, \sigma)$ reachable in $P$, we have $(f(tr), f(th), f(\sigma))$ is reachable in $f(P)$.

**Proof.** We prove the proposition by induction on the number $n$ of transitions leading to a state $(tr, th, \sigma)$.

- Base case ($n = 0$): We have $tr = \epsilon$. Hence $f(tr) = \epsilon$. Moreover, for all $i \in \text{dom}(th)$, there exists $R \in \mathcal{R}_P$ such that $th(i) = \text{inst}_i(P(R))$. Hence we also have $f(th)(i) = f(th(i)) = \text{inst}_i(f(P)(R))$. Since $\text{dom}(\sigma) = V^\mathcal{D}_P$, we have $\text{dom}(f(\sigma)) = V^\mathcal{D}_f(P)$. Thus $(f(tr), f(th), f(\sigma))$ 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

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

(124)

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

67
• If \( r = SEND \) then there exists \( i \in TID \) and \( R \in \mathcal{R}_P \) such that

\[
\begin{align*}
    th'(i) &= (R, \text{snd}(pt) \cdot tl) \\
    th &= th'[i \mapsto (R, tl)] \\
    tr &= tr' \cdot (i, \text{snd}(pt))
\end{align*}
\]

(125)

By (125) we have

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

(126)

By (126) and rule SEND we have

\[
(f(tr'), f(th'), f(\sigma)) \rightarrow (f(tr), f(th), f(\sigma))
\]

Together with (124), this yields \((f(tr), f(th), f(\sigma))\) is reachable in \( f(P) \).

• If \( r = RECV \) then there exists \( i \in TID \) and \( R \in \mathcal{R}_P \) such that

\[
\begin{align*}
    th'(i) &= (R, \text{rcv}(pt) \cdot tl) \\
    IK(tr')\sigma \cup IK_0 \vdash pt\sigma \\
    tr &= tr' \cdot (i, \text{rcv}(pt)) \\
    th &= th'[i \mapsto tl]
\end{align*}
\]

(127)

By (127) we have

\[
\begin{align*}
    f(th')(i) &= (R, \text{rcv}(f(pt)) \cdot f(tl)) \\
    f(tr) &= f(tr') \cdot (i, \text{rcv}(f(pt))) \\
    f(th) &= f(th')[i \mapsto (R, f(tl))]
\end{align*}
\]

(128)

By (124), it is enough to show that

\[
(f(tr'), f(th'), f(\sigma)) \rightarrow (f(tr), f(th), f(\sigma))
\]

By (127) and Lemma 30 we have

\[
IK(f(tr'))f(\sigma) \cup IK_0' \vdash f(pt)f(\sigma)
\]

(129)

By (129), (128) and rule RECV we have

\[
(f(tr'), f(th'), f(\sigma)) \rightarrow (f(tr), f(th), f(\sigma))
\]

This completes the proof of the proposition. \( \square \)

We next show attack preservation for relaying transformation.
**Attack preservation** Relaying transformation also preserves attacks. Interestingly, the result holds for all attacks and protocols.

**Theorem 6 (Attack preservation for relaying).** We assume that

- $P$ is a protocol,
- $\mathcal{X}$ is the independent set of $P$,
- $f$ be a relaying transformation on $\mathcal{M}_P$ with respect to $\mathcal{X} \cup \mathcal{X}^\sharp$,
- $IK_0 \subseteq IK_0'$,
- $\phi \in \mathcal{L}_P$ is closed, and
- $\mathcal{X} \cap \text{vars}(\phi) = \emptyset$.

Then we have, for all states $(\text{tr}, \text{th}, \sigma)$ reachable in $P$,

- $((f(\text{tr}), f(\text{th}), f(\sigma)))$ is reachable in $f(P)$ and
- $(\text{tr}, \text{th}, \sigma) \not\models \phi$ implies $(f(\text{tr}), f(\text{th}), f(\sigma)) \not\models f(\phi)$.

**Proof.** Let $(\text{tr}, \text{th}, \sigma)$ be a reachable state of $P$. The first conjunct, namely that the state $(f(\text{tr}), f(\text{th}), f(\sigma))$ is reachable in $f(P)$, follows from Proposition 10. To show the second conjunct, we prove the following statement by induction on the structure of $\phi$:

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

We consider the following cases:

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

  $$(\text{tr}, \text{th}, \sigma, \vartheta) \models i = j$$
  $$\Leftrightarrow \vartheta(i) = \vartheta(j)$$
  $$\Leftrightarrow (f(\text{tr}), f(\text{th}), f(\sigma), \vartheta) \models f(i = j)$$

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

  $$(\text{tr}, \text{th}, \sigma, \vartheta) \models m = m'$$
  $$\Leftrightarrow m\sigma = m'\sigma$$
  $$\Leftrightarrow f(m\sigma) = f(m'\sigma)$$
  $$\Leftrightarrow f(m)f(\sigma) = f(m')f(\sigma)$$
  by Corollary 4
  $$\Leftrightarrow f(m)f(\sigma) = f(m')f(\sigma)$$
  since $\mathcal{X} \cap \text{vars}(\phi) = \emptyset$,
  and by Corollary 4
  $$\Leftrightarrow (f(\text{tr}), f(\text{th}), f(\sigma), \vartheta) \models f(m) = f(m')$$

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

  $$(\text{tr}, \text{th}, \sigma, \vartheta) \models \text{role}(i, R)$$
  $$\Leftrightarrow \exists \text{seq} \in \text{Event}. \ \text{th}(\vartheta(i)) = (R, \text{seq})$$
  $$\Leftrightarrow \exists \text{seq} \in \text{Event}. \ f(\text{th})(\vartheta(i)) = (R, f(\text{seq}))$$
  $f$ is the identity on $\mathcal{R}_P^\sharp$
  $$\Leftrightarrow (f(\text{tr}), f(\text{th}), f(\sigma), \vartheta) \models \text{role}(i, f(R))$$
This completes the proof of the theorem.

Here are the inductive cases:

- \( \phi \equiv honest(i, R) \) or \( \phi \equiv \neg honest(i, R) \).
  
  \[(tr, th, \sigma, \emptyset) \models honest(i, R) \]
  \[\iff R_{\emptyset(i)} \sigma \in A_H \]
  \[\iff f(R_{\emptyset(i)} \sigma) \in A_H \quad \text{f is the identity on } A \]
  \[\iff f(R_{\emptyset(i)}) f(\sigma) \in A_H \quad \text{by Theorem 2} \]
  \[\iff (f(tr), f(th), f(\sigma), \emptyset) \models honest(i, f(R)) \]

- \( \phi \equiv \text{steps}(i, s(m)) \) or \( \phi \equiv \neg \text{steps}(i, s(m)) \), where \( s \in \{\text{snd, rcv}\} \).
  
  \[(tr, th, \sigma, \emptyset) \models \text{steps}(i, s(m)) \]
  \[\iff (\emptyset(i), s(m)) \in tr \]
  \[\iff (\emptyset(i), s(f(m))) \in f(tr) \quad \text{since } X \cap \text{vars}(\phi) = \emptyset, \quad \text{and by Corollary 4} \]
  \[\iff (f(tr), f(th), f(\sigma), \emptyset) \models \text{steps}(i, s(m)) \]

- \( \phi \equiv (i, s(m)) \prec (j, s'(m')) \) or \( \phi \equiv \neg ((i, s(m)) \prec (j, s'(m'))) \), where \( s, s' \in \{\text{snd, rcv}\} \).
  
  \[(tr, th, \sigma, \emptyset) \models (i, s(m)) \prec (j, s'(m')) \]
  \[\iff (i, s(m)) \prec_{tr} (j, s'(m')) \]
  \[\iff (i, s(f(m))) \prec_{f(tr)} (j, s'(f(m'))) \quad \text{since } X \cap \text{vars}(\phi) = \emptyset, \quad \text{and by Corollary 4} \]
  \[\iff (f(tr), f(th), f(\sigma), \emptyset) \models (i, s(f(m))) \prec (j, s'(f(m'))) \]

- \( \phi \equiv \text{secret}(m) \).
  
  \[(tr, th, \sigma, \emptyset) \not\models \text{secret}(m) \]
  \[\iff IK(tr) \sigma \cup IK_0 \vdash m \sigma \]
  \[\implies IK(f(tr)) f(\sigma) \cup IK'_0 \vdash f(m) f(\sigma) \quad \text{by Lemma 30} \]
  \[\iff (f(tr), f(th), f(\sigma), \emptyset) \not\models \text{secret}(f(m)) \]

Here are the inductive cases:

- \( \phi = \phi_1 \land \phi_2 \).
  
  \[(tr, th, \sigma, \emptyset) \not\models \phi \]
  \[\iff (tr, th, \sigma, \emptyset) \not\models \phi_i \quad \text{for some } i \in \{1, 2\} \]
  \[\implies (f(tr), f(th), f(\sigma), \emptyset) \not\models f(\phi_i) \quad \text{by induction hypothesis} \]
  \[\iff (f(tr), f(th), f(\sigma), \emptyset) \not\models f(\phi) \]

- \( \phi = \phi_1 \lor \phi_2 \). Similar to case for disjunction.

- \( \phi = \forall i. \phi' \).
  
  \[(tr, th, \sigma, \emptyset) \not\models \forall i. \phi' \]
  \[\iff (tr, th, \sigma, \emptyset[i \mapsto \text{tid}]) \not\models \phi' \quad \text{for some } \text{tid} \in \text{dom}(th) \]
  \[\implies (f(tr), f(th), f(\sigma), \emptyset[i \mapsto \text{tid}]) \not\models f(\phi') \quad \text{by induction hypothesis} \]
  \[\iff (f(tr), f(th), f(\sigma), \emptyset) \not\models f(\forall i. \phi') \]

- \( \phi = \exists i. \phi' \). Similar to case for existential quantifier.

This completes the proof of the theorem. \( \square \)