Abstractions for Security Protocol Verification

Binh Thanh Nguyen and Christoph Sprenger

Institute of Information Security
Department of Computer Science, ETH Zurich, Switzerland

Abstract. We present a large class of security protocol abstractions with the aim of improving the scope and efficiency of verification tools. We propose typed abstractions, which transform a term's structure based on its type, and untyped abstractions, which remove atomic messages, variables, and redundant terms. Our theory improves on previous work by supporting a useful subclass of *shallow* subterm-convergent rewrite theories, user-defined types, and untyped variables to cover type flaw attacks. We prove soundness results for an expressive property language that includes secrecy and authentication. Applying our abstractions to realistic IETF protocol models, we achieve dramatic speedups and extend the scope of several modern security protocol analyzers.

1 Introduction

Security protocols play a central role in today's networked applications. Past experience has amply shown that informal arguments justifying the security of such protocols are insufficient. This makes security protocols prime candidates for formal verification. In the last two decades, research in formal security protocol verification has made enormous progress, which is reflected in many state-of-the-art tools including AVANTSSAR [1], ProVerif [6], Maude-NPA [14], Scyther [10], and Tamarin [21]. These tools can verify small to medium-sized protocols in a few seconds or less, sometimes for an unbounded number of sessions. Despite this success, they can still be challenged when verifying real-world protocols such as those defined in standards and deployed on the internet (e.g., TLS, IKE, and ISO/IEC 9798). Such protocols typically have messages with numerous fields, support many alternatives (e.g., cryptographic setups), and may be composed from more basic protocols (e.g., IKEv2-EAP).

Abstraction [7] is a standard technique to over-approximate complex systems by simpler ones for verification. Sound abstractions preserve counterexamples (or attacks in security terms) from concrete to abstracted systems. In the context of security protocols, abstractions are extensively used. Here, we only mention a few examples. First, the Dolev-Yao model is a standard (not necessarily sound) abstraction of cryptography. Second, many tools use abstractions to map the verification problem into the formalism of an efficient solver or reasoner. We call these *back-end* abstractions. For example, ProVerif [6] translates models in the applied pi calculus to a set of Horn clauses, SATMC [4] reduces protocol verification to SAT solving, and Paulson [24] models protocols as inductively defined trace sets. Finally, some abstractions aim at speeding up automated analysis by simplifying protocols within a given protocol model before feeding them to verifiers [18,22]. Our work belongs to this class of *front-end* abstractions.

Extending Hui and Lowe's work [18], we proposed in [22] a rich class of protocol abstractions and proved its soundness for a wide range of security properties. We used a type system to uniformly transform all terms of a given type (e.g., a pattern in a protocol role and its instances during execution) whereas [18] only covers ground terms. Our work [22] exhibits several limitations: (1) the theory is limited to the free algebra over a fixed signature; (2) all variables have strict (possibly structured) types, hence we cannot precisely model ticket forwarding or Diffie-Hellman exchanges. While the type system enables fine-grained control over abstractions (e.g., by discerning different nonces), it may eliminate realistic attacks such as type flaw attacks; (3) some soundness conditions involving quantifiers are hard to check in practice; and (4) it presents few experimental results for a single tool (SATMC) using abstractions that are crafted manually.

In this work, we address all the limitations above. First, we work with a useful subclass of *shallow* subterm-convergent rewrite theories modulo a set of axioms to model cryptographic operations. Second, we support untyped variables, user-defined types, and subtyping. User-defined types enable the grouping of similar atomic types (e.g., session keys) and adjusting the granularity of matching in message abstraction. Third, we have separated the removal of variables, atomic messages, and redundancies (new untyped abstractions) from the transformation of the message structure (typed abstractions). This simplifies the specifications and soundness proof of typed abstractions. Fourth, we provide effectively checkable syntactic criteria for the conditions of the soundness theorem. Finally, we extended Scyther [10] with fully automated support for our abstraction methodology. We validated our approach on an extensive set of realistic case studies drawn from the IKEv1, IKEv2, ISO/IEC 9798, and PANA-AKA standard proposals. Our abstractions result in very substantial performance gains. We have also obtained positive results for several other state-of-the-art verifiers (ProVerif, CL-Atse, OFMC, and SATMC) with manually produced abstractions.

Example: The IKEv2-mac protocol The Internet Key Exchange (IKE) family of protocols is part of the IPsec protocol suite for securing Internet Protocol (IP) communication. IKE establishes a shared key, which is later used for securing IP packets, realizes mutual authentication, and offers identity protection as an option. Its first version (IKEv1) dates back to 1998 [17]. The second version (IKEv2) [20] significantly simplifies the first one. However, the protocols in this family are still complex and contain a large number of fields.

Concrete protocol. As our running example, we present a member of the IKEv2 family, called IKEv2-mac (or $\mathsf{IKE}_{\mathsf{m}}$ for short), which sets up a session key using a Diffie-Hellman (DH) key exchange, provides mutual authentication based on MACs, and also offers identity protection. We use Cremers' models of IKE [11] as a basis for our presentation and experiments (see Section 4.2). Our starting point is the following concrete $\mathsf{IKE_m}$ protocol between an initiator A and a responder B.

$$\begin{aligned}
&\mathsf{IKE_m}(1).\;\; A \to B : \mathit{SPIa}, o, \mathit{sA1}, g^x, \mathit{Na}\\
&\mathsf{IKE_m}(2).\;\; B \to A : \mathit{SPIa}, \mathit{SPIb}, \mathit{sA1}, g^y, \mathit{Nb}\\
&\mathsf{IKE_m}(3).\;\; A \to B : \mathit{SPIa}, \mathit{SPIb}, \{\!|A, B, \mathit{AUTHa}, \mathit{sA2}, \mathit{tSa}, \mathit{tSb}|\!\}_{\mathit{SK}}\\
&\mathsf{IKE_m}(4).\;\; B \to A : \mathit{SPIa}, \mathit{SPIb}, \{\!|B, \mathit{AUTHb}, \mathit{sA2}, \mathit{tSa}, \mathit{tSb}|\!\}_{\mathit{SK}}
\end{aligned}$$

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

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

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

We consider the following security properties: (P1) the secrecy of the DH key g^{xy}, which implies the secrecy of SK, and (P2) mutual non-injective agreement on the nonces Na and Nb and the DH half-keys g^x and g^y.

Abstraction. Our theory supports the construction of abstract models by removing inessential fields and operations. For example, in $\mathsf{IKE_m}$ we can remove: (i) the symmetric encryptions with the session key SK; then (ii) all atomic top-level fields except Na and Nb; (iii) all fields of SK except the DH key g^{xy}; and (iv) from the authenticators: the fields SPIa, SPIb, and $\mathit{sA1}$ and the application of prf including the agent names underneath. The resulting protocol is $\mathsf{IKE^2_m}$:

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

where $\mathit{SK} = \mathsf{kdf}(g^{xy})$ and $\mathit{AUTHa} = \mathsf{mac}(\mathsf{sh}(A,B), o, g^x, \mathit{Na}, \mathit{Nb}, \mathit{SK})$ for role A and $\mathit{AUTHb} = \mathsf{mac}(\mathsf{sh}(B,A), g^y, \mathit{Nb}, \mathit{Na}, \mathit{SK})$ for role B.

Scyther verifies the properties (P1) and (P2) in 8.7s on the concrete and in 1.7s on an automatically generated abstract protocol (which is less intuitive than the one presented here). Our soundness results imply that the original protocol $\mathsf{IKE_m}$ also enjoys these properties. We chose the protocol $\mathsf{IKE_m}$ as running example for its relative simplicity compared to the other protocols in our case studies. In many of our experiments (Section 4.2), our abstractions (i) result in much more substantial speedups, or (ii) enable the successful unbounded verification of a protocol where it times out or exhausts memory on the original protocol.

2 Security protocol model

We define a term algebra $\mathcal{T}_{\Sigma}(V)$ over a signature Σ and a set of variables V in the standard way. Let Σ^n denote the symbols of arity n. We call the elements of Σ^0

atoms and write $\Sigma^{\geq 1}$ for the set of proper function symbols. For a fixed $\Sigma^{\geq 1}$, we will vary Σ^0 to generate different sets of terms, denoted by $\mathcal{T}(V, \Sigma^0)$, including terms in protocol roles, network messages, and types. We write $subs(t)$ for the set of subterms of t and define the size of t by $|t| = |subs(t)|$. We also define $vars(t) = subs(t) \cap V$. If $vars(t) = \emptyset$ then t is called *ground*. We denote the top-level symbol of a (non-variable) term t by $top(t)$ and the set of its symbols in $\Sigma^{\geq 1}$ by $ct(t)$. A position is a sequence of natural numbers. We denote the subterm of t at position p with $t|_p$ and write $t[u]_p$ for the term obtained by replacing $t|_p$ at position p by u. We also partition Σ into sets of public and private symbols, denoted by Σ_{pub} and Σ_{pri}. We assume Σ_{pub} includes pairing $\langle \cdot, \cdot \rangle$ which associates to the right, e.g., $\langle t, u, v \rangle = \langle t, \langle u, v \rangle \rangle$. We usually write, e.g., $\{|t, u, v|\}_k$ rather than $\{|\langle t, u, v \rangle|\}_k$. We define the *splitting* function by $split(\langle t, u \rangle) = split(t) \cup split(u)$ on pairs and $split(t) = \{t\}$ on other terms t. We call the elements of $split(t)$ the *fields* of t. For $n \in \mathbb{N}$, $\widetilde{n}$ denotes $\{1, \ldots, n\}$.

The set of *message terms* is $\mathcal{M} = \mathcal{T}(\mathcal{V}, \mathcal{A} \cup \mathcal{F} \cup \mathcal{C})$, where $\mathcal{V}$, $\mathcal{A}$, $\mathcal{F}$, and $\mathcal{C}$ are pairwise disjoint infinite sets of variables, agents, fresh values, and constants.

2.1 Type system

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

We assume that all variables have an atomic type, i.e., $\mathcal{V} = \{\mathcal{V}_\tau\}_{\tau \in \mathcal{Y}_{at}}$ is a family of disjoint infinite sets of variables. Let $\Gamma \colon \mathcal{V} \to \mathcal{Y}_{at}$ be such that $\Gamma(X) = \tau$ if and only if $X \in \mathcal{V}_\tau$. We extend Γ to atoms by defining $\Gamma(a) = \alpha$, $\Gamma(n) = \beta_n$, and $\Gamma(c) = \gamma_c$ for $a \in \mathcal{A}$, $n \in \mathcal{F}$, and $c \in \mathcal{C}$, and then homomorphically to all terms $t \in \mathcal{M}$. We call $\tau = \Gamma(t)$ the *type of* t and sometimes also write $t : \tau$.

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

$$\frac{\tau \in \mathcal{Y}}{\tau \preccurlyeq msg}\ \mathrm{S}(msg) \qquad \frac{\tau_1 \preccurlyeq_0 \tau_2}{\tau_1 \preccurlyeq \tau_2}\ \mathrm{S}(\preccurlyeq_0) \qquad \frac{\tau_1 \preccurlyeq \tau_1' \quad \cdots \quad \tau_n \preccurlyeq \tau_n'}{c(\tau_1, \ldots, \tau_n) \preccurlyeq c(\tau_1', \ldots, \tau_n')}\ \mathrm{S}(c \in \Sigma^n)$$

Every type is a subtype of msg by the first rule. The second rule embeds a user-defined *atomic subtyping relation* $\preccurlyeq_0 \subseteq (\mathcal{Y}_{at} \setminus \{msg\}) \times \mathcal{Y}_0$, which relates atomic types (except msg) to user-defined atomic types in $\mathcal{Y}_0$. For simplicity, we require that $\preccurlyeq_0$ is a partial function. The third rule ensures that subtyping is preserved by all symbols. The set of subtypes of τ is $\tau{\downarrow} = \{\tau' \in \mathcal{Y} \mid \tau' \preccurlyeq \tau\}$.

2.2 Equational theories

An *equation* over a signature Σ is an unordered pair $\{s, t\}$, written $s \simeq t$, where $s, t \in \mathcal{T}_\Sigma(\mathcal{V}_{msg})$. An *equation presentation* $\mathcal{E} = (\Sigma, E)$ consists of a signature Σ and a set E of equations over Σ. The *equational theory* induced by $\mathcal{E}$ is the smallest Σ-congruence, written $=_E$, containing all instances of equations in E. We often identify $\mathcal{E}$ with the induced equational theory.

A *rewrite rule* is an oriented pair $l \to r$, where $vars(r) \subseteq vars(l) \subseteq \mathcal{V}_{msg}$. A *rewrite theory* is a triple $\mathcal{R} = (\Sigma, Ax, R)$ where Σ is a signature, Ax a set of Σ-equations, and R a set of rewrite rules. The rewriting relation $\to_{R,Ax}$ on $\mathcal{T}_\Sigma(V)$ is defined by $t \to_{R,Ax} t'$ iff there exists a non-variable position p in t, a rule $l \to r \in R$, and a substitution σ such that $t|_p =_{Ax} l\sigma$ and $t' = t[r\sigma]_p$. If $t \to^*_{R,Ax} t'$ and t' is irreducible, we call t' *R,Ax-normal* and also say that t' is a *normal form* of t. Under suitable termination, confluence, and coherence conditions (see [19] for definitions), one can decompose an equational theory (Σ, E) into a rewrite theory (Σ, Ax, R) where $Ax \subseteq E$ and, for all terms $t, u \in \mathcal{T}_\Sigma(V)$, we have $t =_E u$ iff $t{\downarrow_{R,Ax}} =_{Ax} u{\downarrow_{R,Ax}}$. Here, $t{\downarrow_{R,Ax}}$ denotes any normal form of t. In this paper, we work with decomposable equational theories.

A rewriting theory R is *subterm-convergent* if it is convergent and, for each $l \to r \in R$, r is either a proper subterm of l or ground and in normal form with respect to R. For our soundness result, we consider the subclass $\mathcal{S}$ of subterm-convergent rewrite theories where each rule in R has one of the following forms.

– (R1): $d(c(x_1, \ldots, x_n, t), u) \to x_j$, where $c, d \in \Sigma_{\mathsf{pub}}$, t, u are terms, $j \in \widetilde{n}$, and $x_1, \ldots, x_n$ are pairwise distinct variables with $x_i \notin vars(t, u)$ for all $i \in \widetilde{n}$.
– (R2): $d(c(x_1, \ldots, x_n)) \to x_j$, where $c, d \in \Sigma_{\mathsf{pub}}$, $j \in \widetilde{n}$, and $x_1, \ldots, x_n$ are pairwise distinct variables.
– (R3): $c(x_1, \ldots, x_n) \to x_j$ where $c \in \Sigma_{\mathsf{pub}}$, x_j is a variable with $j \in \widetilde{n}$, and x_i is a variable or an atom for all $i \in \widetilde{n}$.
– (R4): $l \to a$ for a constant a.

Intuitively, the first three forms enable different types of projection of a term's arguments. Rules R1 and R2 apply a destructor d to extract one of c's arguments. In rule R1 the destructor has two arguments. The terms t and u can be seen a pair of matching keys required to extract x_j. Rule R3 uses no destructor. Finally, R4 models rewriting a term to a constant. Since the rules (R1-R3) have limited depth, we call the class $\mathcal{S}$ of rewrite theories *shallow subterm-convergent*.

We also introduce a condition on the equations Ax of the rewrite theory.

Definition 1. *A rewrite theory* (Σ, Ax, R) *is* well-formed *if for all* $\{s, t\} \in Ax$, *we have (i) neither* s *nor* t *is a pair and (ii)* $top(s) = top(t)$.

We only consider equational theories that can be decomposed into a shallow subterm-convergent, well-formed rewrite theory. These are adequate to model many well-known cryptographic primitives as illustrated by the examples below.

Example 1. We model the protocols of our case studies (see Sections 1 and 4) in the rewrite theory $\mathcal{R}_{cs} = (\Sigma_{cs}, Ax_{cs}, R_{cs})$ where

$$\Sigma_{cs} = \{\mathsf{sh}, \mathsf{pk}, \mathsf{pri}, \mathsf{prf}, \mathsf{kdf}, \mathsf{mac}, \langle\cdot,\cdot\rangle, \pi_1, \pi_2, \{|\cdot|\}_\cdot, \{|\cdot|\}_\cdot^{-1}, \{\cdot\}_\cdot, \{\cdot\}_\cdot^{-1}, [\cdot]_\cdot, \mathsf{ver}\} \cup \Sigma^0_{cs}$$

contains function symbols for: shared, public, and private long-term keys (where $\Sigma_{\mathsf{pri}} = \{\mathsf{sh}, \mathsf{pri}\}$); hash functions prf, kdf, and mac; pairs and projections; symmetric and asymmetric encryption and decryption; and signing and verification.

The set of atoms Σ^0_{cs} is specified later. The set R_{cs} consists of rewrite rules for projections (type R2) and for decryption and signature verification (type R1):

$$\begin{array}{lll} \pi_1(\langle X,Y\rangle) \to X & \{\!|\{\!|X|\!\}_K|\!\}^{-1}_K \to X & \mathsf{ver}([X]_{\mathsf{pri}(K)}, \mathsf{pk}(K)) \to X \\ \pi_2(\langle X,Y\rangle) \to Y & \{\{X\}_{\mathsf{pk}(K)}\}^{-1}_{\mathsf{pri}(K)} \to X & \end{array}$$

We have two equations in Ax_{cs}, namely, $\mathsf{exp}(\mathsf{exp}(g,X),Y) \simeq \mathsf{exp}(\mathsf{exp}(g,Y),X)$ to model Diffie-Hellman key exchange and $\mathsf{sh}(X,Y) \simeq \mathsf{sh}(Y,X)$.

Example 2. The theory of XOR is given by the following rewrite system where the rules are of types R2, R3 and R4. The rightmost rule ensures coherence [19].

$$\begin{array}{rll} X \oplus Y \simeq Y \oplus X & X \oplus 0 \to X & X \oplus X \oplus Y \to Y \\ (X \oplus Y) \oplus Z \simeq X \oplus (Y \oplus Z) & X \oplus X \to 0 & \end{array}$$

For our theoretical development, we consider an arbitrary but fixed shallow subterm-convergent and well-formed rewrite theory (Σ, Ax, R) that includes the function symbols and rewrite rules for pairing and projections.

We denote by $dom(g)$ and $ran(g)$ the domain and range of a function g. We now define well-typed substitutions, which respect subtyping.

Definition 2 (Well-typed substitutions). *A substitution θ is* well-typed *if $\Gamma((X\theta){\downarrow_{R,Ax}}) \preccurlyeq \Gamma(X)$ for all $X \in dom(\theta)$.*

2.3 Protocols

For a set of terms T, we define the set of events $Evt(T) = \{\mathsf{snd}(t), \mathsf{rcv}(t) \mid t \in T\}$ and $term(ev(t)) = t$ for event $ev(t)$. A *role* is a sequence of events from $Evt(\mathcal{M})$.

Definition 3 (Protocol). *A* protocol *is a function $P : \mathcal{V}_\alpha \rightharpoonup Evt(\mathcal{M})^*$ mapping agent variables to roles. Let $\mathcal{M}_P = term(ran(P))$ be the set of* protocol terms *appearing in the roles of P, and let $\mathcal{V}_P$, $\mathcal{A}_P$, $\mathcal{F}_P$, and $\mathcal{C}_P$ denote the sets of variables, agents, fresh values, and constants in $\mathcal{M}_P$.*

Example 3 ($\mathsf{IKE_m}$ protocol). We formalize the $\mathsf{IKE_m}$ protocol from Section 1 in the rewrite theory of Example 1 as follows, using upper-case (lower-case) identifiers for variables (atoms). The atoms Σ^0_{cs} are composed of constants $C = \{g, o, sA1, sA2, tSa, tSb\}$ and fresh values $F = \{na, nb, x, y, sPIa, sPIb\}$. The variables and their types are $A, B : \alpha$, $Ga, Gb : msg$, $SPIa, SPIb, Na, Nb : nonce$ where $nonce$ is a user-defined type that satisfies $\beta_n \preccurlyeq_0 nonce$ for all $n \in F$. We show here the initiator role A. The responder role B is dual.

$$\begin{aligned} \mathsf{IKE_m}(A) = {} & \mathsf{snd}(sPIa, o, sA1, \mathsf{exp}(g,x), na) \cdot \mathsf{rcv}(sPIa, SPIb, sA1, Gb, Nb) \cdot \\ & \mathsf{snd}(sPIa, SPIb, \{\!|A, B, AUTHaa, sA2, tSa, tSb|\!\}_{SKa}) \cdot \\ & \mathsf{rcv}(sPIa, SPIb, \{\!|B, AUTHba, sA2, tSa, tSb|\!\}_{SKa}) \end{aligned}$$

where the terms $SKa = \mathsf{kdf}(na, Nb, \mathsf{exp}(Gb, x), sPIa, SPIb)$ and

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

represent the initiator A's view of the session key and of the authenticators.

$$\frac{u \in T}{T \vdash_E u}\ \mathsf{Ax} \qquad \frac{T \vdash_E t' \quad t' =_E t}{T \vdash_E t}\ \mathsf{Eq} \qquad \frac{T \vdash_E t_1 \quad \cdots \quad T \vdash_E t_n}{T \vdash_E f(t_1, \ldots, t_n)}\ \mathsf{Comp}\ (f \in \Sigma^{\geq 1}_{\mathsf{pub}})$$

Fig. 1. Intruder deduction rules (where $\Sigma^{\geq 1}_{\mathsf{pub}} = \Sigma^{\geq 1} \cap \Sigma_{\mathsf{pub}}$)

2.4 Operational semantics

Let *TID* be a countably infinite set of thread identifiers. When we instantiate a role into a thread for execution, we mark its variables and fresh values with the thread identifier i. We define the instantiation $t^{\#i}$ of a term t for $i \in \mathit{TID}$ as the term where every variable or fresh value u is replaced by u^i. Constants and agents remain unchanged. Instantiation does not affect the type of a term.

We define by $T^\sharp = \{t^{\#i} \mid t \in T \wedge i \in \mathit{TID}\}$ the set of instantiations of terms in a set T and abbreviate $T^\flat = T \cup T^\sharp$. For example, $\mathcal{M}^\sharp$ is the set of instantiated message terms, which we will use to instantiate roles into threads. We define the set of *network messages* exchanged during protocol execution by $\mathcal{N} = \mathcal{T}(\mathcal{V}^\sharp, \mathcal{A} \cup \mathcal{F}^\sharp \cup \mathcal{F}^\bullet \cup \mathcal{C})$, where $\mathcal{F}^\bullet = \{n^\bullet_k \mid n \in \mathcal{F} \wedge k \in \mathbb{N}\}$ is the set of attacker-generated fresh values. Note that $\mathcal{M}^\sharp \subseteq \mathcal{N}$. We abbreviate $\mathcal{T} = \mathcal{M} \cup \mathcal{N}$.

We use a Dolev-Yao attacker model parametrized by an equational theory E. Its judgements are of the form $T \vdash_E t$ meaning that the intruder can derive term t from the set of terms T. The derivable judgements are defined in a standard way by the three deduction rules in Figure 1.

We define a transition system with states (tr, th, σ), where

- tr is a *trace* consisting of a sequence of pairs of thread identifiers and events,
- $th : \mathit{TID} \rightharpoonup \mathit{dom}(P) \times \mathit{Evt}(\mathcal{M}^\sharp_P)^*$ are *threads* executing role instances, and
- $\sigma : \mathcal{V}^\sharp \rightharpoonup \mathcal{N}$ is a well-typed ground substitution from instantiated protocol variables to network messages such that $\mathcal{V}^\sharp_P \subseteq \mathit{dom}(\sigma)$.

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

The set Init_P of initial states of protocol P contains all (ϵ, th, σ) satisfying

$$\forall i \in \mathit{dom}(th).\ \exists R \in \mathit{dom}(P).\ th(i) = (R, P(R)^{\#i})$$

where all terms in the respective protocol roles are instantiated. The substitution σ is chosen non-deterministically in the initial state.

The rules in Figure 2 define the transitions. In both rules, the first premise states that a send or receive event heads thread i's role. This event is removed and added together with the thread identifier i to the trace tr. The substitution σ remains unchanged. The second premise of *RECV* requires that the network message $t\sigma$ matching the term t in the receive event is derivable from the intruder's (ground) knowledge $\mathit{IK}(tr)\sigma \cup \mathit{IK}_0$. Here, $\mathit{IK}(tr)$ denotes the (symbolic) intruder

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

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

Fig. 2. Operational semantics

knowledge derived from a trace tr as the set of terms in the send events on tr, i.e., $IK(tr) = \{t \mid \exists i.\ (i, \mathsf{snd}(t)) \in tr\}$ and IK_0 denotes the intruder's (ground) initial knowledge. We assume $\mathcal{A} \cup \mathcal{C} \cup \mathcal{F}^\bullet \subseteq IK_0$ and IK_0 is R,Ax-normal. Note that the *SEND* rule implicitly updates this intruder knowledge.

2.5 Property language

We use the same specification language as in [22] to express secrecy and authentication properties. Hence, we only sketch some of its elements and give examples. There are atomic formulas to express equality ($t = u$), the secrecy of a term ($secret(t)$), the occurrence of an event e by thread i in the trace ($steps(i, e)$), that thread i executes role R, and the honesty of other agents in the view of a thread i. Quantification is allowed over thread identifier variables. To achieve attack preservation, the predicate $secret(t)$ may occur only positively.

Example 4 (Properties of $\mathsf{IKE_m}$*).* We express the secrecy of the Diffie-Hellman key $\mathsf{exp}(Gb, x)$ for role A of the protocol $\mathsf{IKE_m}$ of Example 3 as follows.

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

where $t_4 = \langle sPIa, SPIb, \{\!|B, AUTHba, sA2, tSa, tSb|\!\}_{SKa}\rangle$ and $honest(j, [A, B])$ means that A and B are honest. We formalize non-injective agreement of A with B on the nonces na and nb and the DH half-keys $\mathsf{exp}(g, x)$ and $\mathsf{exp}(g, y)$ by

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

3 Security protocols abstractions

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

Typed abstractions transform a term's structure by reordering or removing fields and by splitting or removing cryptographic operations. The same transformations are applied to all terms of a given type and its subtypes.

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

Our main results are soundness theorems for these abstractions. They ensure that any attack on a given property of the original protocol translates to an attack on the abstracted protocol. As we will see, these results hold under certain conditions on the protocol and the property. Here, we focus on typed abstractions, but we will also briefly introduce the untyped ones (see [23] for details).

3.1 Typed protocol abstractions

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

Syntax Let $\mathcal{W} = \{\mathcal{W}_\tau\}_{\tau \in \mathcal{Y}}$ be a family of *pattern variables* disjoint from $\mathcal{V}$. We define the set of *patterns* by $\mathcal{P} = \mathcal{T}(\mathcal{W}, \emptyset)$. A pattern $p \in \mathcal{P}$ is called *linear* if each (pattern) variable occurs at most once in p. We extend the typing function Γ to patterns by setting $\Gamma(X) = \tau$ if and only if $X \in \mathcal{W}_\tau$ and then lifting it homomorphically to all patterns. Our typed message abstractions are instances of the following recursive function specifications.

Definition 4. *A* function specification $F_f = (f, E_f)$ *consists of an unary function symbol* $f \notin \Sigma^1$ *and a list of equations*

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

where each $p_i \in \mathcal{P}$ *is a linear pattern such that* $u_i \in \mathcal{T}_{\Sigma^{\geq 1} \cup \{f\}}(vars(p_i))$ *for all* $i \in \widetilde{n}$*, i.e.,* u_i *consists of variables from* p_i *and function symbols from* $\Sigma^{\geq 1} \cup \{f\}$*.*

We use vectors (lists) of terms $\overline{t} = [t_1, \ldots, t_n]$ for $n > 0$. We define $set(\overline{t}) = \{t_1, \ldots, t_n\}$ and $\widehat{f}(\overline{t}) = \langle f(t_1), \ldots, f(t_n) \rangle$, the elementwise application of a function f to a vector where the result is converted to a tuple (with the convention $\langle t \rangle = t$). We extend *split* to vectors by $split(\overline{t}) = split(set(\overline{t}))$. We define three sets of function symbols occurring in R and Ax as follows.

$$\begin{aligned}
\mathsf{C}_{\mathsf{R}} &= \{c \mid d(c(x_1, \ldots, x_n, t), u) \to x_j \in R\} \\
\mathsf{C}_{\mathsf{Key}} &= \bigcup \{ct(t) \cup ct(u) \mid d(c(x_1, \ldots, x_n, t), u) \to x_j \in R\} \\
\mathsf{C}_{\mathsf{Ax}} &= \bigcup \{ct(s) \cup ct(t) \mid \{s, t\} \in Ax\}
\end{aligned}$$

The function $pp(c)$ returns the set of extractable indices of a function symbol c, i.e., $pp(c) = \{j \mid d(c(x_1, \ldots, x_n, t), u) \to x_j \in R$ or $d(c(x_1, \ldots, x_n)) \to x_j \in R\}$.

Definition 5 (Typed abstraction). *A function specification* $F_f = (f, E_f)$ *is a* typed abstraction *if each equation in* E_f *has the form*

$$f(c(p_1, \ldots, p_n)) = \langle e_1, \ldots, e_d \rangle$$

where for each $i \in \widetilde{d}$ *we have either*

(a) $e_i = f(q)$ *such that* $q \in split(p_j)$ *for some* $j \in \widetilde{n}$*, or*
(b) $e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))$ *such that* $set(\overline{q_j}) \subseteq split(p_j)$ *for all* $j \in \widetilde{n}$*,* $c \neq \langle \cdot, \cdot \rangle$*, and* $c \in \mathsf{C_R}$ *implies* $\overline{q_n} = [p_n]$*, i.e.,* $\widehat{f}(\overline{q_n}) = f(p_n)$*.*

Moreover, we require (i) for all $j \in pp(c)$ *we have* $split(p_j) \subseteq Q_j$ *where*

$$Q_j = \bigcup \{set(\overline{q_j}) \mid \exists i \in \widetilde{d}.\, e_i = c(\widehat{f}(\overline{q_1}), \ldots, \widehat{f}(\overline{q_n}))\} \cup \{q \mid \exists i \in \widetilde{d}.\, e_i = f(q)\}.$$

and (ii) if $c \in \mathsf{C_{Ax}} \cup \mathsf{C_{Key}}$ *then* $p_i = X_i : msg$ *for all* $i \in \widetilde{n}$*,* $d = 1$ *and* $e_1 = c(f(X_1), \ldots, f(X_n))$ *is an instance of (b); we say* F_f is homomorphic for c.

Intuitively, the abstractions can only weaken the cryptographic protection of terms, but never strengthen it. Each equation in E_f maps a term with top-level symbol c to a tuple whose components have the form (a) or (b). Form (a) allows us to pull fields out of the scope of c, hence removing c's protection. Using form (b) we can reorder or remove fields in each argument of c. Form (b) is subject to two conditions. First, we disallow this form for pairs to obtain the simple shape $f(\langle p_1, p_2 \rangle) = \widehat{f}(\overline{q})$. Second, we cannot permit the reordering or removal of fields in key positions, i.e., in the last argument of $c \in \mathsf{C_R}$. Moreover, by point (i), all fields of extractable arguments, i.e., elements of $split(p_j)$ for $j \in pp(c)$, must be present in some e_i and point (ii) requires that the abstraction is homomorphic for function symbols c occurring in axioms and in keys ($c \in \mathsf{C_{Ax}} \cup \mathsf{C_{Key}}$).

Example 5. We present a typed abstraction $F_f = (f, E_f)$ illustrating a representative selection of the possible message transformations. Suppose $X : \gamma_c$, $Y : nonce$, and $Z, U, V : msg$ and let E_f consists of the following three equations:

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

The patterns' types filter the matching terms: X and Y only match the constant c and a nonce, respectively. The first equation swaps the first two fields in n-tuples for $n \geq 3$. The second one splits a kdf hash into two, removing the field V. The last equation splits an encryption: the pair $\langle f(X), f(Y) \rangle$ and $f(Z)$ are encrypted separately with the key $f(U)$ and $f(Y)$ is pulled out of the encryption. Note that by condition (i) of Definition 5, we cannot directly remove plaintext fields from encryptions. To achieve this, we pull such fields out of encryptions to the top-level. This may require a combination of several abstractions if there are multiple layers of cryptographic protection. At the top-level, the fields are no longer protected and can be removed using untyped abstractions. In Section 4.1, we will discuss our heuristics to determine sequences of abstractions automatically.

Semantics The semantics of a typed abstraction F_f is given by the Haskell-style functional program f (Program 1).[1] To ensure totality, we use the extended function specification $(f, E_f^+) = (f, E_f \cdot E_f^0)$, where $f(g(Z_1, \ldots, Z_n)) =$

[1] We are overloading the symbol f here, but no confusion should arise.

```
fun $f(t)$ = case $t$ of
    $\|_{(f(p)=u)\in E_f^+}\ p \mid \Gamma(t) \preccurlyeq \Gamma(p) \Rightarrow u$
```

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

$g(f(Z_1), \ldots, f(Z_n)) \in E_f^0$ for each $g \in \Sigma^n$ with $n \geq 1$ such that $Z_i : msg$ for all $i \in \widetilde{n}$, and $f(Z) = Z$ with $Z : msg$ is the last clause in E_f^0. We assume E_f and E_f^0 do not share variables. The **case** statement has a clause

$$p \mid \Gamma(t) \preccurlyeq \Gamma(p) \Rightarrow u$$

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

We extend f to events, event sequences, and traces by applying f to the terms they contain and to substitutions and protocols by applying f to the terms in their range. Similarly, we extend f to formulas ϕ of our property language by applying f to all terms occurring in ϕ.

Finding abstractions Finding abstractions is fully automated by our tool using a heuristic that we will describe in Section 4.1. However, the resulting abstractions can be counterintuitive. Therefore, we present here a simplified strategy that we apply to our running example below: We start by identifying the terms that appear in the *secret*$(\cdot)$ predicates and equations of the desired properties. Then we determine the cryptographic operations that are essential to achieve these properties and try to remove all other terms and operations.

Example 6 (from $\mathsf{IKE_m}$ *to* $\mathsf{IKE_m^1}$*).* In order to preserve the secrecy of the DH key $\mathsf{exp}(\mathsf{exp}(g,x),y)$ and the agreement on na, nb, $\mathsf{exp}(g,x)$, and $\mathsf{exp}(g,y)$, we have to keep either the mac or the symmetric encryption with SK (see Examples 3 and 4). We want to remove as many other fields and operations as possible (e.g., prf). We choose to remove the encryption as this allows us to later remove additional fields (e.g., $sA2$) using untyped abstractions. We keep o in $AUTHa$ to prevent unifiability with $AUTHb$ and hence potential false negatives. This leads us to the typed abstraction $F_1 = (f_1, E_1)$ where E_1 is defined by the equations

$$\begin{aligned} f_1(\{\!|X,Y|\!\}_Z) &= \langle f_1(X), f_1(Y)\rangle \\ f_1(\mathsf{mac}(X_1,\ldots,X_8)) &= \mathsf{mac}(\widehat{f_1}([X_1,X_3,X_5,X_6,X_7,X_8])) \\ f_1(\mathsf{mac}(Y_1,\ldots,Y_8)) &= \mathsf{mac}(\widehat{f_1}([Y_1,Y_5,Y_6,Y_7,Y_8])) \\ f_1(\mathsf{kdf}(Z_1,\ldots,Z_5)) &= \mathsf{kdf}(f_1(Z_3)) \\ f_1(\mathsf{prf}(U,Z)) &= f_1(U) \end{aligned}$$

(where we omitted the homomorphic clauses for exp and $\langle\cdot,\cdot\rangle$) and $X : \alpha$, $X_3 : \gamma_o$, $Y_3 : nonce$, $Z_3 : \mathsf{exp}(msg, msg)$, $U : \mathsf{kdf}(msg)$ and all remaining pattern variables are of type msg. Applying f_1 to $\mathsf{IKE_m}$ we obtain $\mathsf{IKE^1_m}$. Here is the abstracted initiator role.

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

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

3.2 Soundness of typed abstractions

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

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

These properties will require some assumptions about the protocol P, the formula ϕ, and the abstraction f. Before we formally state the soundness theorem, we will introduce and motivate these assumptions while sketching its proof. For the remainder of this subsection we assume arbitrary but fixed P, ϕ, F_f.

We start with two basic properties of abstractions. The first one, which we call the *substitution property*, states that $f(t\theta) = f(t)f(\theta)$ for well-typed R,Ax-normal substitutions θ. This does not hold in general. For example, suppose E_f contains the clauses $f(h(Y{:}\gamma_c)) = f(X)$ and $f(h(X{:}msg)) = h(f(X))$ in this order. Then the property is violated for $t = h(Z{:}msg)$ and $\theta = [c/Z]$. Thus, we must ensure that t and all its instance $t\theta$ are transformed uniformly, i.e., match the same clauses of E_f. We therefore require that (i) the patterns in E_f must not overlap and (ii) all recursive calls of f on composed terms during the transformation of t are handled by the clauses of E_f, without recourse to the fall-back clauses in E^0_f. This is formalized in the following two definitions where we denote the set of pattern types of a list of equations L by $\Pi(L) = \{\Gamma(p) \mid (f(p) = u) \in L\}$, we define $\Pi_f = \Pi(E_f)$, and let $Rec(F_f, t)$ be the set of terms u such that $f(u)$ is called in the computation of $f(t)$.

Definition 6. *A function specification F_f is* pattern-disjoint *if the types in Π_f are pairwise disjoint, i.e., $\Gamma(p_i)\!\downarrow \cap\, \Gamma(p_j)\!\downarrow = \emptyset$ for all $i, j \in \widetilde{n}$ such that $i \neq j$.*

Definition 7 (Uniform domain). *We define the* uniform domain *of F_f by $udom(F_f) = \{t \in \mathcal{T} \mid \Gamma(Rec(F_f, t)) \subseteq \Pi_f\!\downarrow \cup\, \mathcal{Y}_{at}\}$.*

We will require that the protocol terms $t \in \mathcal{M}_P$ belong to this set, which ensures that their instances $t\theta$ with R,Ax-normal substitutions θ are transformed uniformly. We henceforth assume that F_f is pattern-disjoint. Note that the abstractions defined in Examples 5 and 6 are pattern-disjoint.

Theorem 1 (Substitution property). *Let $t \in udom(F_f)$ and θ be a well-typed and R,Ax-normal substitution. Then $f(t\theta) = f(t)f(\theta)$.*

The second basic property needed in our soundness proof is that abstractions preserve equality modulo E. We decompose this into the preservation of Ax-equality and of rewriting steps. Neither is preserved in general. To ensure this we need the following two definitions.

Definition 8 (R, Ax-closedness). *F_f is* R, Ax-closed *if the following holds: $t =_{Ax} u$ implies $\tau_t \preccurlyeq \tau$ if and only if $\tau_u \preccurlyeq \tau$, for all R,Ax-normal composed terms $t : \tau_t$ and $u : \tau_u$ and all $\tau \in \Pi(E_f^+)$.*

We henceforth assume that F_f is R, Ax-closed. In [23], we present a syntactic criterion for checking this. To achieve the preservation of rewriting steps under abstraction, we must ensure that, for all positions p in t where a rule $l \to r \in R$ is applicable, the redex $t|_p$ in t is transformed into a redex $f(t|_p)$ in $f(t)$ that still Ax-matches l. This is the purpose of the following definition.

Definition 9 (R, Ax-homomorphism). *We say that f is R, Ax-*homomorphic *for a term t if for all non-variable positions p in t and for all rules $l \to r \in R$ such that there exists a well-typed Ax-unifier of $t|_p$ and l, it holds that*

(i) f is homomorphic for all $c \in ct(l)$,
(ii) f is homomorphic for $top(t|_{p'})$ and $top(t|_{p'}) \neq top(l')$ for all strict prefixes p' of p and rewrite rules $l' \to r' \in R$ such that $ct(l')$ is not a singleton.

We define $rdom(F_f)$ to be the set of terms for which f is R, Ax-homomorphic.

Many interesting protocols P satisfy $\mathcal{M}_P \subseteq rdom(F_f)$, including those from our case studies. Since we must also cover redexes arising by instantiating protocol terms $t \in \mathcal{M}_P$, this definition employs Ax-unification rather than Ax-matching. The definition ensures that instantiations with R,Ax-normal substitutions and rewriting steps both preserve the membership of terms in $rdom(F_f)$.

Theorem 2 (Equality preservation). *Let t and u be terms such that $t, u \in rdom(F_f)$. Then $t =_E u$ implies $f(t) =_E f(u)$.*

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

$$T\theta, IK_0 \vdash_E u\theta \;\Rightarrow\; f(T)f(\theta{\downarrow_{R,Ax}}), f(IK_0) \vdash_E f(u)f(\theta{\downarrow_{R,Ax}}). \tag{1}$$

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

Theorem 3 (Deducibility preservation). *Let $T \cup \{t\} \subseteq \mathcal{N}$ be a set of ground network messages such that $\mathcal{C} \subseteq T$ and T is R,Ax-normal. Then $T \vdash_E t$ implies $f(T) \vdash_E f(t{\downarrow_{R,Ax}})$.*

We can now derive (1) by applying Theorems 3, 2 and 1 in this order, combined with applications of rule Eq and a cut property of intruder deduction. Summarizing, reachability preservation (RP) holds for $\mathcal{M}_P \subseteq udom(F_f) \cap rdom(F_f)$.

Attack preservation (AP) We next define and explain the conditions on formulas needed to establish attack preservation. Let

- Sec_ϕ be the set of all terms t that occur in formulas $secret(t)$ in ϕ,
- Eq_ϕ be the set of pairs (t, u) such that the equation $t = u$ occurs in ϕ and $EqTerm_\phi = \{t, u \mid (t, u) \in Eq_\phi\}$ is the set of underlying terms, and
- Evt_ϕ be the set of events occurring in ϕ.

Let Eq^+_ϕ the positively occurring equations in ϕ and similarly for Evt_ϕ.

Definition 10 (Safe formulas). *ϕ is* safe *for P and f if*

(i) $Sec_\phi \cup EqTerm_\phi \subseteq udom(F_f) \cap rdom(F_f)$,
(ii) $f(t\sigma) =_E f(u\sigma)$ *implies* $t\sigma =_E u\sigma$ *for all* $(t, u) \in Eq^+_\phi$ *and for all well-typed R,Ax-normal ground substitutions σ, and*
(iii) $f(t) = f(u)$ *implies* $t = u$*, for all* $e(t) \in Evt^+_\phi$ *and* $e(u) \in Evt(\mathcal{M}_P)$.

Condition (i) requires that F_f is uniform and R, Ax-homomorphic for the terms in secrecy statements and equalities. Condition (ii) expresses the injectivity of the abstraction on the terms in positively occurring equalities. This condition is required to preserve attacks on agreement properties. In other words, it prevents abstractions from fixing attacks on agreement by identifying two terms that differ in the original protocol. In the full version [23], we provide a syntactic criterion to check condition (ii) that avoids the universal quantification over substitutions. Condition (iii) is required for properties involving event orderings and *steps* predicates. It states that the abstraction must not identify an event occurring positively in the property with a distinct protocol event.

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

Theorem 4 (Soundness). *Suppose P, ϕ, and F_f satisfy (i) $f(IK_0) \subseteq IK'_0$, (ii) F_f is pattern-disjoint and R, Ax-closed, (iii) $\mathcal{M}_P \subseteq udom(F_f) \cap rdom(F_f)$, and ϕ is safe for P and f. Then, for all states (tr, th, σ) reachable in P, we have*

1. $(f(tr), f(th), f(\sigma{\downarrow_{R,Ax}}))$ *is a reachable state of* $f(P)$*, and*
2. $(tr, th, \sigma) \not\models \phi$ *implies* $(f(tr), f(th), f(\sigma{\downarrow_{R,Ax}})) \not\models f(\phi)$.

3.3 Untyped abstractions

Typed abstractions offer a wide range of possibilities to transform cryptographic operations including subterm removal, splitting, and pulling fields outside a cryptographic operation. We complement these abstractions with two kinds of *untyped abstractions* that allow us to remove (1) unprotected atoms and variables of any type and (2) redundancy in the form of intruder-derivable terms. Untyped protocol abstractions are functions $g : \mathcal{T} \to \mathcal{T} \cup \{\mathsf{nil}\}$ where messages to be removed are mapped to nil. We remove events with nil arguments from the roles. Due to lack of space, we only sketch the definitions and give an example here. Full details and soundness results can be found in [23].

Atom/variable removal The *removal abstraction* $rem_T : \mathcal{T} \to \mathcal{T} \cup \{\mathsf{nil}\}$ for a set T of atoms or variables is defined by

- $rem_T(u) = \mathsf{nil}$ if $u \in T^\flat$,
- $rem_T(\langle t_1, t_2 \rangle) = \begin{cases} rem_T(t_i) & \text{if } rem_T(t_{3-i}) = \mathsf{nil} \text{ for some } i \in \widetilde{2} \\ \langle rem_T(t_1), rem_T(t_2) \rangle & \text{otherwise} \end{cases}$
- $rem_T(t) = t$ for all other terms.

In order to preserve attacks, we have to restrict the removal of atoms and variables from a protocol term t to fields $u \in split(t)$ that appear only unprotected (clear) in t, i.e., such that $u \notin subs(t) \setminus split(t)$.

Example 7 ($\mathsf{IKE}^1_\mathsf{m}$ to $\mathsf{IKE}^2_\mathsf{m}$). We use atom/variable removal to simplify the protocol $\mathsf{IKE}^1_\mathsf{m}$. First, we recall the specification of role A of $\mathsf{IKE}^1_\mathsf{m}$.

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

We remove the role names A and B, the constants o, $sA1$, $sA2$, tSa, tSb, the fresh value $sPIa$, and the variable $SPIb$ using an atom/variable removal abstraction. The result is the protocol $\mathsf{IKE}^2_\mathsf{m}$ whose initiator role is defined as follows.

$$S_{\mathsf{IKE}^2_\mathsf{m}}(A) = \mathsf{snd}(\mathsf{exp}(g, x), na) \cdot \mathsf{rcv}(Gb, Nb) \cdot \mathsf{snd}(AUTHaa) \cdot \mathsf{rcv}(AUTHba)$$

We also apply the typed abstraction from Example 6 and the untyped abstraction here to the properties ϕ_s and ϕ_a of Example 4. These only affect the events in the *steps* predicates. The relevant soundness conditions are satisfied.

Redundancy removal A redundancy removal abstraction rd enables the elimination of redundancies within each role of a protocol. Intuitively, a protocol term t appearing in a role r can be abstracted to $rd(t)$ if t and $rd(t)$ are derivable from each other under the intruder knowledge T containing the terms preceding t in r and the initial knowledge IK_0. For example, we can simplify $r = \mathsf{snd}(t) \cdot \mathsf{rcv}(\langle t, u \rangle)$ to $\mathsf{snd}(t) \cdot \mathsf{rcv}(u)$. In contrast to atom/variable removal, redundancy removal can also remove composed terms. It is therefore a very effective ingredient for automatic abstraction, which we describe next.

4 Implementation and experimental results

We have implemented our abstraction methodology for the Scyther tool and tested it on a variety of complex protocols, mainly stemming from the IKE and ISO/IEC 9798 families. Scyther is an efficient verifier for security protocols. It supports verification for both a bounded and an unbounded number of threads. Protocols are specified by a set of linear role scripts. It also supports user-defined types. These features match our setting very well.

4.1 Abstraction heuristics

Our tool computes a series of successively more abstract protocols. Each abstraction step consists of a typed abstraction followed by a redundancy and an atom/variable removal abstraction. A heuristic guides the automatic generation of the typed abstractions. These abstractions may be partially user-specified.

Central to our heuristic are the (sub)terms of Sec_ϕ and EqTerm_ϕ for a given property ϕ, which we call *essential terms*. The heuristic assigns *security labels*, c for confidentiality and a for authenticity, to cryptographic primitives as their intended security guarantees. These labels are inherited by subterms. Concretely, we label symmetric encryptions and MACs with c and a, asymmetric encryptions and hashes with c, and signatures with a. Based on this labeling, we decide which fields are pulled outside of or removed from the topmost cryptographic operations. The main criterion is that these transformations must preserve the following labeling properties of each essential term t: the presence of an a label on *some* occurrence of t and of c labels on *all* occurrences of t. The successive abstractions work from the outside to the inside of the original protocol's terms. The untyped abstractions simply remove all inessential top-level fields.

Example 8. We can simplify the term $\{\!|B, \mathit{AUTHba}, \mathit{sA2}, \mathit{tSa}, \mathit{tSb}|\!\}_{\mathit{SKa}}$ where $\mathit{AUTHba} = \mathsf{mac}(\mathsf{sh}(A, B), \mathit{sPIa}, \mathit{SPIb}, \mathit{sA1}, \underline{\mathit{Gb}}, \underline{\mathit{Nb}}, \underline{\mathit{na}}, \mathsf{prf}(\mathit{SKa}, B))$ of the $\mathsf{IKE_m}$ protocol from Example 3 in two successive abstraction steps as follows.

$$\begin{aligned} \{\!|B, \mathit{AUTHba}, \mathit{sA2}, \mathit{tSa}, \mathit{tSb}|\!\}_{\mathit{SKa}} &\mapsto \langle B, \mathit{AUTHba}, \mathit{sA2}, \mathit{tSa}, \mathit{tSb}\rangle \\ \mathit{AUTHba} &\mapsto \mathsf{mac}(\mathsf{sh}(A, B), \underline{\mathit{Gb}}, \underline{\mathit{Nb}}, \underline{\mathit{na}}, \mathsf{prf}(\mathit{SKa}, B)) \end{aligned}$$

In the first step, we pull the whole plaintext out of the encryption since the security labels of essential terms (underlined) are preserved by the mac. In the second step, we transform AUTHba by keeping essential and removing inessential terms. Note that removing the term $u = \mathsf{prf}(\mathit{SKa}, B)$ or pulling it out of the mac would not preserve authenticity for the essential term x inside SKa. In a further step, we can simplify u by deleting inessential subterms and dropping prf.

Our abstractions are sound, but not complete. Therefore, we may encounter false negatives, i.e., spurious attacks. We carefully try to avoid these, for instance, by checking that abstractions do not introduce new pairs of unifiable terms. We currently do not check automatically whether an attack is spurious. Whenever an attack on a protocol P is found, we proceed to analyze (only) the failed properties on the next more concrete protocol in the series of abstractions.

4.2 Experimental results

We have validated the effectiveness of our abstractions on 22 members of the IKE and ISO/IEC 9798 protocol families and on the PANA-AKA protocol [3]. We verify these protocols using five tools based on four different techniques: Scyther [10], CL-Atse [26], OFMC [5], SATMC [4], and ProVerif [6]. Only Scyther and ProVerif support verification of an unbounded number of threads. Due to

protocol	No	S	A	W	N	3	4	5	6	7	8	∞
IKEv1-pk-a22	1	✓			✓	18.48	82.93	249.55	554.09	1006.04	1734.85	TO
						0.83	1.26	2.08	3.47	5.96	10.28	TO
IKEv2-eap	5	✓			✓	TO	TO	TO	TO	TO	TO	TO
						78.35	798.44	4212.71	20911.20	TO	TO	TO
IKEv2-mac	5	✓			✓	1.85	4.91	6.72	8.07	8.42	8.49	8.70
						0.62	1.77	1.83	1.73	1.73	1.80	1.74
IKEv2-mactosig	4	✓			✓	11.65	141.37	1075.46	7440.81	TO	TO	TO
						2.89	12.38	24.54	38.68	53.36	65.07	77.68
IKEv2-sigtomac	5	✓			✓	6.15	33.19	65.05	115.34	204.93	206.45	237.34
						3.59	12.72	28.44	44.44	55.11	66.97	67.15
IKEv1-pk-m	2				×	48.62	269.92	507.40	869.23	16254.80	TO	TO
						0.04	0.05	0.05	0.05	0.05	0.05	TO
IKEv1-pk-m2	2				✓/×	18.26	274.87	4438.72	TO	TO	TO	TO
						1.48	7.79	32.75	110.32	339.93	963.08	TO
IKEv1-sig-m	2				×	0.34	0.45	0.45	0.45	0.45	0.46	0.44
						0.05	0.05	0.05	0.06	0.05	0.05	0.06
IKEv1-sig-m-perlman	2				×	2.86	13.99	40.78	67.83	72.08	72.15	109.03
						0.05	0.05	0.05	0.05	0.05	0.05	0.05
ISO/IEC 9798-2-5	1	✓				0.78	8.96	73.87	564.67	4214.22	TO	TO
						0.07	0.11	0.12	0.11	0.11	0.11	0.11
ISO/IEC 9798-2-6	1	✓				0.57	3.74	18.42	67.01	196.30	488.04	21278.58
						0.05	0.04	0.05	0.05	0.05	0.05	0.05
ISO/IEC 9798-3-6-1	2		✓		✓	43.08	802.95	8903.70	ME	ME	ME	ME
						0.13	0.18	0.19	0.19	0.19	0.19	0.19
ISO/IEC 9798-3-6-2	1		✓		✓	2.74	8.67	19.56	33.91	52.51	69.48	90.04
						0.12	0.15	0.15	0.15	0.15	0.15	0.15
ISO/IEC 9798-3-7-1	2		✓		✓	40.43	740.47	7483.36	16631.42	ME	ME	ME
						0.13	0.18	0.19	0.19	0.19	0.19	0.19
ISO/IEC 9798-3-7-2	1		✓		✓	2.38	7.71	16.68	26.99	35.06	49.49	TO
						0.22	0.32	0.33	0.33	0.33	0.33	0.33
PANA-AKA	5	✓	✓	✓	✓	5769.53	TO	TO	TO	TO	TO	TO
						0.10	0.10	0.10	0.10	0.10	0.10	0.10

Table 1. Experimental results. The time is in seconds. **No**: Number of abstractions. Properties: **S**ecrecy, **A**liveness, **W**eak agreement, and **N**on-injective agreement.

lack of space, we present only a selection of 16 experimental results for Scyther (Table 1) and refer to the full version [23] for a complete account. Our models of the IKE and ISO/IEC 9798 protocols are based on Cremers' [8,9]. Since Scyther uses a fixed signature with standard cryptographic primitives and no equational theories, the IKE models approximate the DH theory by oracle roles.

We mark verified properties by ✓ and falsified ones by ×. An entry ✓/× means the property holds for one role but not for the other. Each row consists of two lines, corresponding to the analysis time without (line 1) and with (line 2) abstraction for 3-8 or unboundedly many (∞) threads. The times were measured on a cluster of 12-core AMD Opteron 6174 processors with 64 GB RAM each. They include computing the abstractions (4-20 ms) and the verification itself.

Verification For 8 of the 12 original protocols that are verified, an unbounded verification attempt results in a timeout (TO = 8h cpu time) or memory exhaustion (ME). In 6 of these, our abstractions enabled a verification in less than 0.4 seconds and in one case in 78 seconds. However, for the first two protocols, we still get a timeout. For the large majority of the bounded verification tasks, we significantly push the bound on the number of threads and achieve massive speedups. For example, our abstractions enable the verification of the complex nested protocols IKEv2-eap and PANA-AKA. Scyther verifies an abstraction of IKEv2-eap for up to 6 threads and, more strikingly, completes the unbounded verification of the simplified PANA-AKA in under 0.1 seconds whereas it can handle only 4 threads of the original. We also achieve dramatic speedups for many other protocols, most notably for IKEv1-pk-a22, ISO/IEC 9798-2-6, and ISO/IEC 9798-3-6-2. Moreover, the verification time for many abstracted protocols increases much more slowly than for their originals. We obtain almost constant verification times for the six ISO/IEC 9798 protocols, whereas the time significantly increases on some originals, e.g., for ISO/IEC 9798-3-6-1. For a few protocols, e.g., IKEv2-sigtomac and IKEv2-mac, the speedup is more modest.

Falsification For rows marked by $\times$, the second line corresponds to falsification time for the most abstract model, which is much faster than on the original one. For example, for 8 threads of the IKEv1-pk-m protocol, we reduce falsification time from a timeout to 0.05 seconds. In the unbounded case, the speedup factors are 7 for IKEv1-sig-m and 2180 for IKEv1-sig-m-perlman. A manual analysis of the abstract attacks shows that none of them is spurious, suggesting that our measures to prevent them are effective. We expect that fast automatic detection of spurious attacks is feasible and will affect performance only negligibly.

Combination For the IKEv1-pk-m2 protocol, the tool verifies non-injective agreement for one role and falsifies it for the other one. Surprisingly, we obtain a remarkable speedup even though the analysis of this protocol is done three times (for two abstract and the original models). Our abstractions push the feasibility bound from 5 to 8 threads. As the property is verified very quickly for one role on the most abstract model, it needs to be analyzed only for the other role at lower abstraction levels. This explains the remarkable speedups we obtain and therefore illustrates an advantage of our abstraction mechanism in this case.

5 Related work and conclusions

Hui and Lowe [18] define several kinds of abstractions similar to ours with the aim of improving the performance of the CASPER/FDR verifier. They establish soundness only for ground messages and encryption with atomic keys. We work in a more general model, cover additional properties, and treat the non-trivial issue of abstracting the open terms in protocol specifications. Other works [25,13,12] also propose a set of syntactic transformations, however without formally establishing their soundness. Using our results, we can, for instance, justify the soundness of the refinements in [13, Section 3.3]. Guttman [16,15] studies the preservation of security properties for a rich class of protocol transformations in the strand space model. His approach to property preservation is based on the simulation of protocol analysis steps instead of execution steps. Each such step explains the origin of a message. He does not have a syntactic soundness check.

In this work, we propose a set of syntactic protocol transformations that allows us to abstract realistic protocols and capture a large class of attacks. Unlike previous work [22,18], our theory and soundness results accommodate equational theories, untyped variables, user-defined types, and subtyping. These features allow us to accurately model protocols, capture type-flaw attacks, and adapt to different verification tools, e.g., those supporting equational theories such as ProVerif and CL-atse. We have extended Scyther with an abstraction module, which we validated it on various IKE and ISO/IEC 9798 protocols. We also tested our technique (with manually produced abstractions) on ProVerif, CL-atse, OFMC, and SATMC. Our experiments clearly show that modern protocol verifiers can substantially benefit from our abstractions, which often either enable previously unfeasible verification tasks or lead to dramatic speedups.

Our abstraction tool does not check for spurious attacks. We plan to add this functionality to complete the automatic abstraction-refinement process. We are also interested in generalizing the tool and supporting more protocol verifiers.

Acknowledgements We thank Mathieu Turuani and Michael Rusinowitch for our discussions on the topic of this paper, Cas Cremers for his help with Scyther, David Basin, Ognjen Maric, and Ralf Sasse for their careful proof-reading, and the anonymous reviewers for their useful feedback. This work is partially supported by the EU FP7-ICT-2009 Project No. 256980, NESSoS: Network of Excellence on Engineering Secure Future Internet Software Services and Systems.

References

1. A. Armando et al.: The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In: Flanagan, C., König, B. (eds.) TACAS. Lecture Notes in Computer Science, vol. 7214, pp. 267–282. Springer (2012)
2. Arapinis, M., Duflot, M.: Bounding messages for free in security protocols. In: Arvind, V., Prasad, S. (eds.) FSTTCS. Lecture Notes in Computer Science, vol. 4855, pp. 376–387. Springer (2007)
3. Arkko, J., Haverinen, H.: RFC 4187: Extensible authentication protocol method for 3rd generation authentication and key agreement (EAP-AKA) (2006), `http://www.ietf.org/rfc/rfc4187`
4. Armando, A., Compagna, L.: SAT-based model-checking for security protocols analysis. International Journal of Information Security 7(1), 3–32 (2008)
5. Basin, D.A., Mödersheim, S., Viganò, L.: OFMC: A symbolic model checker for security protocols. Int. J. Inf. Sec. 4(3), 181–208 (2005)
6. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: CSFW. pp. 82–96. IEEE Computer Society (2001)
7. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Graham, R.M., Harrison, M.A., Sethi, R. (eds.) POPL. pp. 238–252. ACM (1977)

8. Cremers, C.: IKEv1 and IKEv2 protocol suites (2011), `https://github.com/cascremers/scyther/tree/master/gui/Protocols/IKE`
9. Cremers, C.: ISO/IEC 9798 authentication protocols (2012), `https://github.com/cascremers/scyther/tree/master/gui/Protocols/ISO-9798`
10. Cremers, C.J.F.: The Scyther tool: Verification, falsification, and analysis of security protocols. In: Gupta, A., Malik, S. (eds.) CAV. Lecture Notes in Computer Science, vol. 5123, pp. 414–418. Springer (2008)
11. Cremers, C.J.F.: Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In: Atluri, V., Díaz, C. (eds.) ESORICS. Lecture Notes in Computer Science, vol. 6879, pp. 315–334. Springer (2011)
12. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: Abstraction and refinement in protocol derivation. In: Proc. 17th IEEE Computer Security Foundations Workshop (CSFW) (2004)
13. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security 13, 423–482 (2005)
14. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: Cryptographic protocol analysis modulo equational properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD. Lecture Notes in Computer Science, vol. 5705, pp. 1–50. Springer (2007)
15. Guttman, J.D.: Transformations between cryptographic protocols. In: Degano, P., Viganò, L. (eds.) ARSPA-WITS. LNCS, vol. 5511, pp. 107–123. Springer (2009)
16. Guttman, J.D.: Security goals and protocol transformations. In: Theory of Security and Applications (TOSCA). LNCS, vol. 6993. Springer (2011)
17. Harkins, D., Carrel, D.: The Internet Key Exchange (IKE). IETF RFC 2409 (Proposed Standard) (Nov 1998), `http://www.ietf.org/rfc/rfc2409.txt`
18. Hui, M.L., Lowe, G.: Fault-preserving simplifying transformations for security protocols. Journal of Computer Security 9(1/2), 3–46 (2001)
19. Jouannaud, J., Kirchner, H.: Completion of a set of rules modulo a set of equations. SIAM J. Comput. 15(4), 1155–1194 (1986)
20. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P.: Internet Key Exchange Protocol Version 2 (IKEv2). IETF RFC 5996 (September 2010), `http://tools.ietf.org/html/rfc5996`
21. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN prover for the symbolic analysis of security protocols. In: Sharygina, N., Veith, H. (eds.) CAV. Lecture Notes in Computer Science, vol. 8044, pp. 696–701. Springer (2013)
22. Nguyen, B.T., Sprenger, C.: Sound security protocol transformations. In: Basin, D.A., Mitchell, J.C. (eds.) POST. Lecture Notes in Computer Science, vol. 7796, pp. 83–104. Springer (2013)
23. Nguyen, B.T., Sprenger, C.: Abstractions for security protocol verification. Tech. rep., Department of Computer Science, ETH Zurich (January 2015), `http://dx.doi.org/10.3929/ethz-a-010347780`
24. Paulson, L.: The inductive approach to verifying cryptographic protocols. J. Computer Security 6, 85–128 (1998)
25. Pavlovic, D., Meadows, C.: Deriving secrecy in key establishment protocols. In: Proc. 11th European Symposium on Research in Computer Security (ESORICS). pp. 384–403 (2006)
26. Turuani, M.: The CL-Atse protocol analyser. In: Pfenning, F. (ed.) RTA. Lecture Notes in Computer Science, vol. 4098, pp. 277–286. Springer (2006)