Deciding Security for a Fragment of ASLan (Extended Version)*

Sebastian Mödersheim

DTU Informatics, Denmark, samo@imm.dtu.dk

IMM-Technical Report-2012-06
Third edition, April 11, 2018

Revised version of April 11, 2018

As part of our latest research [12] we have discovered a flaw in this paper that we previously went unnoticed because of a too sketch proof. In fact, since many details of these proofs are so subtle, we are currently formalizing them with Isabelle/HOL.

In the following you find the revised version where all changes are highlighted using change bars (as this paragraph); also we give as footnotes the original version and the explanation what is wrong with it.

Summary of the changes:

- In the original paper, the requirements on the intruder rules are too lax, i.e., they allow for intruder rules where the result would not hold. This has been changed to the set allowed by [12].
- The original version allows for variabels of composed terms in predicates, so they can actually end up in inequalities during symbolic execution, which is not sound in this generality. We follow [12] to restrict the use of composed-type variables. In fact, it seems we can relax these conditions some more, we are currently investigating this.

This should not be a significant restriction in practice, as the counter examples are not what a typical TASLan specification should contain.

To the best of our knowledge, this is now all correct, and we of course invite everybody to double check the results and notify us of anything that seems unclear or wrong.

* The author thanks Luca Viganò, Alberto Calvi, Marco Rocchetto, and the anonymous reviewers (of the conference version) for many helpful comments. The research presented in this paper has been partially supported by MT-LAB, a VKR Centre of Excellence for the Modelling of Information Technology.

Abstract. ASLan is the input language of the verification tools of the AVANTSSAR platform, and an extension of the AVISPA Intermediate Format IF. One of ASLan's core features over IF is to integrate a transition system with Horn clauses that are evaluated at every state. This allows for modeling many common situations in security such as the interaction between the workflow of a system with its access control policies.
While even the transition relation is undecidable for ASLan in general, we show the security problem is decidable for a large and useful fragment that we call TASLan, as long as we bound the number of steps of honest participants. The restriction of TASLan is that all messages and predicates must be in a certain sense unambiguous in their interpretation, excluding "type-confusions" similar to some tagging results for security protocols.

1 Introduction

It is well-understood how to automatically verify small security protocols that consist of the exchange of a few messages. Less well understood is the automated verification of complex distributed systems that we see today in practice, where the logic of a component comprises more than a few message exchanges. An example is a web server that maintains a database (e.g. of keys, of electronic orders, or of electronic applications). This database may be accessed or modified by different transactions the server can perform. These transactions themselves may be embedded into a larger workflow of a company that runs the server, e.g., how employees of the company process requests posted by customers via the server. Finally, there may be access control policies specifying who is allowed to perform which actions or has access to certain information.

Modeling such complex systems requires an expressive specification language. We consider in this paper the AVANTSSAR [2] Specification Language ASLan [4] that was designed in exactly this spirit—to model complex systems like the ones just sketched. At the core, an ASLan specification describes an infinite-state transition system where every state is a set of (ground, first-order) predicates that express, for instance, the local state of honest agents (or uncorrupted components), what messages are known to the intruder, the state of databases shared by agents, or facts related to the security goals such as which messages are supposed to be secret. The transition relation is expressed by *set rewriting rules* (similar to multi-set rewriting [9], only the repetition of predicates does not make a difference). Additionally, ASLan allows for negative conditions in rules.

A powerful feature of ASLan on top of this transition system is the specification of Horn clauses over state predicates. These Horn clauses are evaluated locally in every state and give rise to a set of *implicit* consequences. These consequences are used in matching the next transition rule. For instance, we may express a Horn theory that models access control rules such as "If file F belongs to group G and A is a member of G then A has access to F." or "If A is a deputy of B, then A has all access rights that B has." Membership in a group, or being a deputy are predicates that may change upon state transitions. The Horn

clauses thus allow us to formulate immediate consequences of a state, and after each transition, they are automatically updated. Vice-versa, the Horn clauses may themselves be used as conditions in a transition rule, e.g. A may perform a certain action only in a state from which the necessary access rights can be derived by the Horn clauses. More generally, the Horn theories allow for modeling all kinds of internal computations, expressed as such immediate consequences.

Even though we have chosen here the particular language ASLan, we believe that the concepts that we deal with are of general relevance for the modeling of complex systems, in particular the immediate evaluation of consequences in every state of a state transition system. (As an example, recall that the common Dolev-Yao model of an intruder is represented as the least closure of the messages that the intruder has seen under a set of deduction rules.)

The expressivity of ASLan however comes at a price for automated verification: since first-order Horn clauses allow for logic programming, the transition relation is in general undecidable. In fact it is common that specification languages give rise to undecidable problems, and the challenge is to find fragments for which feasible decision procedures are possible.

Contributions We first review the syntax and semantics of ASLan and make some conceptual simplifications. We exclude at this point some features of ASLan that are in our opinion less essential, but difficult to handle; we briefly discuss how to (partially) support them in section 5.

Next, we define the fragment TASLan forbidding certain kinds of ambiguities in the formats of messages and predicates. TASLan requires, that all messages are annotated with an intended type such that all messages, and their non-variable subterms, that occur in the specification have no unifier unless they have the same intended type. We also extend this restrictions to other predicates. We then show that a TASLan specification has an attack iff it has a well-typed attack, so restriction to a typed model is no restriction for TASLan.

This result is in the spirit of several tagging results [11, 7, 1], and generalizes them: we do not require a particular way to avoid ambiguities (such as tagging) and do not limit ourselves to particular analysis technique (such as ProVerif); and, most importantly, our result works for full TASLan, including non-atomic keys, negative conditions (such as those needed for authentication), and the additional Horn theories.

This result allows for a number of simplifications of the model, in particular bounding the size of terms without restriction. We develop a decision procedure for bounded-length TASLan: given a bound l, can we reach an attack state in l steps or less? This procedure is generalizing the popular constraint-based approach that we refer to as the *lazy intruder* [15, 16, 6]. In fact, this procedure is part of our argument for the typing result on TASLan. For the theoretical closure, we show that the problem whether an attack is reachable in at most l steps for a TASLan specification is NEXPTIME complete.

Organization In section 2 we review the ASLan syntax and semantics. In section 3 we introduce a symbolic transition system that is the basis for the later

$D ::=$	Declarations	$F ::=$	Facts
$c : \beta$	Constant Symbol	P	Predicate
$X : \tau$	Variable Symbol	$t_1 = t_2$	Equality
$f : \boldsymbol{\tau}$	Function Symbol	$\exists \boldsymbol{X} : F$	Existential quantification
$p : \mathsf{pred}\ \boldsymbol{\tau}$	Predicate Symbol	$L ::=$	Literals
$\tau ::=$	Types	F	Fact
β	Basic type	$\neg F$	Negated Fact
$f(\boldsymbol{\tau})$	Composed type	$S ::=$	States
$\mathsf{untyped}$	Untyped	$\hat{P}$	Conjunction of predicates
$s, t ::=$	Terms	$R ::=$	Transition Rules
c	Constant	$\hat{L} \Rightarrow[\boldsymbol{X}]\Rightarrow S$	
X	Variable	$H ::=$	Horn Clauses
$f(\boldsymbol{t})$	Composed Terms	$\forall \boldsymbol{X} : S \to P$	
$P ::=$	Predicates	$\mathcal{P} ::=$	ASLan Specification
$p(\boldsymbol{t})$		$(\boldsymbol{D}, S, \boldsymbol{R}, \boldsymbol{H})$	

Table 1. Syntax of ASLan

decision procedure. In section 4 we introduce the fragment TASLan and give the decision procedure for bounded length traces. From this procedure we also derive our typing result and conclude with the result on the complexity. In section 5 we briefly discuss aspects of ASLan that we have excluded. We conclude with a discussion of related work in section 6.

2 ASLan

Syntax Table 1 shows the syntax of ASLan (where we have left out some features that are in our opinion less crucial, and support for which we discuss in section 5). We use the following conventions: we introduce syntactic categories by $C ::=$, where the symbol C represents our notation of elements of that category. Each following line represents one alternative for that category. Further, we write $\boldsymbol{v}$ for a vector $v_1, \ldots, v_n$ (where the lengths n of the vectors may be 0, and different vectors may have different lengths). Similarly, we write $\hat{\phi}$ for a conjunction of the form $\phi_1 \wedge \ldots \wedge \phi_n$.

Example 1. To illustrate the concepts of ASLan, we give a toy example in Table 2. Note that in this example we use a notational convention of ASLan that we do not enforce in the treatment of this paper: constant, function and predicate symbols are identifiers that start with a lower-case letter, while variable symbols are all identifiers that start with an upper-case letter. In this example we specify as Horn clauses the access control example from the introduction, together with an initial state and two transition rules. The first transition rule is applicable if A is a member of group $G1$ and A is not the deputy of anybody. Upon the transition, we generate a fresh value of type *gid*—in the rule referred to by the variable $G2$. Then A will be a member of $G2$ (actually the only member so far).

Declarations:

mem : pred $(agent, gid)$	own : pred (gid, fid)
$deputy$: pred $(agent, agent)$	xs : pred $(agent, fid)$
$attack$: pred $()$	$A, B, a, b : agent$
$G, G1, G2, g1, g2 : gid$	$F, F1, F2, f1, f2 : fid$

Initial State:

$mem(a, g1) \wedge mem(b, g2) \wedge own(g1, f1) \wedge own(g2, f2)$

Transition Rules:

$mem(A, G1) \wedge \neg\exists B : deputy(A, B) \Rightarrow[G2]\Rightarrow mem(A, G2)$

$xs(A, F1) \wedge xs(A, F2) \wedge own(A, G1) \wedge own(A, G2) \wedge G1 \neq G2 \Rightarrow attack()$

Horn clauses:

$mem(A, G) \wedge own(G, F) \rightarrow xs(A, F)$

$deputy(A, B) \wedge xs(B, F) \rightarrow xs(A, F)$

Table 2. Toy example of an ASLan specification

Also the left-hand side predicate $mem(A, G1)$ will no longer hold. The second rule is an example how attack states can be defined. Here, we derive an attack whenever in a state an agent A has access to files $F1$ and $F2$ (note $F1 = F2$ is allowed) that belong to groups $G1$ and $G2$, respectively, where $G1 \neq G2$ is required; thus when A has a the same time access to files of different groups, the specification has an attack. Note that the specification is infinite state as the first transition rule can be applied any number of times. □

Type Declarations The declarations section of an ASLan specification is by default only an annotation of intentions of the modeler; we do *not* assume that an intruder always sends well-typed messages, and our semantics will thus be ignoring the type declarations by default. The declarations give a means to statically type-check a specification (i.e., checking in the behavior of honest agents the typing is consistent) and later are relevant for our typing result.

A particularity of our type system is that for functions we do not allow the specification of a return type—the resulting type is always a composed type as follows. If f is declared as a function symbol of type $\tau_1, \ldots, \tau_n$ and $t_1 : \tau_1, \ldots, t_n : \tau_n$ are terms of the appropriate types, then $f(t_1, \ldots, t_n) : f(\tau_1, \ldots, \tau_n)$. Thus the type of a term reflects its composition, and only atomic terms can be of a basic type. The only way to escape this tight typing system is using the "type" untyped. Let Γ be a mapping from all declared symbols to a type. We require that every symbol that occurs in the specification has a unique type-definition. We define a general type judgment relation $t : \tau$ (read: t is of type τ) as follows:

$$\frac{}{s : \tau}\ \Gamma(s) = \tau \qquad \frac{t_1 : \tau_1 \quad \ldots \quad t_n : \tau_n}{f(t_1, \ldots, t_n) : f(\tau_1, \ldots, \tau_n)}\ \Gamma(f) = (\tau_1, \ldots, \tau_n)$$

$$\frac{s : \tau}{s : \mathsf{untyped}} \qquad \frac{t_1 : \tau_1 \quad \ldots \quad t_n : \tau_n}{p(t_1, \ldots, t_n) : p(\tau_1, \ldots, \tau_n)}\ \Gamma(p) = (\tau_1, \ldots, \tau_n)$$

We require that all terms and predicates in the specification have a type according to this specifications, and for equation $t_1 = t_2$, t_1 and t_2 have a type in common.

2.1 Further Context Sensitive Properties

We give further conditions about ASLan specifications that are not definable by a context-free grammar. Let $fv(t)$ denote the free variables of t (for terms, predicates, facts, states). Let $Pos(\hat{L})$ denote the positive facts in a conjunction $\hat{L}$ of literals.

- For a rule $\hat{L} =[\boldsymbol{X}]\Rightarrow S$ we require that $fv(\hat{L}) \uplus \boldsymbol{X} \supseteq fv(S)$. Moreover, $fv(Pos(\hat{L})) = fv(\hat{L})$.
- For a Horn clause $H = \forall \boldsymbol{X} : S \rightarrow P$, we require $fv(H) = \emptyset$ and $fv(P) \subseteq fv(S)$.
- The initial state is ground. Together with the previous two conditions, all reachable states are ground (except in the symbolic approach we define later).
- There are two distinguished predicate symbols ik (for intruder knowledge) and attack with $\Gamma(\mathsf{ik}) = (\mathsf{untyped})$ and $\Gamma(\mathsf{attack}) = ()$. Both symbols are *persistent*: they never get deleted on transitions.
- We call a non-persistent predicate *explicit* if it occurs on the right-hand side of a transition rule and *implicit* if it occurs on the right-hand side of a Horn clause. All predicate symbols except ik and attack must be either explicit or implicit. Denote with $PosE(\hat{L})$ the positive explicit predicates of a rule and with $PosI(\hat{L})$ both the positive implicit and the positive persistent predicates.
- Horn clauses in which ik occurs can only have one of the following two forms:
  - Generate: $\forall X_1, \ldots, X_n : \mathsf{ik}(X_1) \wedge \ldots \wedge \mathsf{ik}(X_n) \rightarrow \mathsf{ik}(f(X_1, \ldots, X_n))$
  - Analyze: $\forall \boldsymbol{X} : \mathsf{ik}(f(\boldsymbol{X})) \wedge \mathsf{ik}(t_1) \wedge \ldots \wedge \mathsf{ik}(t_m) \rightarrow \mathsf{ik}(X)$ such that $X \in \boldsymbol{X}$ and $fv(\boldsymbol{t}) \subseteq \boldsymbol{X}$. This means that all analysis rules must apply to an operator f with arbitrary arguments $\boldsymbol{X}$; the result X of the analysis must be directly one of said arguments; the terms that the intruder needs to know for the decryption, namely $t_1, \ldots, t_m$ can only contain variables of $\boldsymbol{X}$ (i.e., that occur in the analyzed term). [1]
- Implicit and persistent predicates (see Section 2.1) cannot occur negatively in the specification.

[1] This condition was in the original version:

> Analyze: $\forall \boldsymbol{X} : \mathsf{ik}(t) \wedge \mathsf{ik}(t_1) \wedge \ldots \wedge \mathsf{ik}(t_n) \rightarrow \mathsf{ik}(s)$ where s and the t_i are proper subterms of t.

This is too permissive on the intruder deduction rules, because in general this may force the intruder to analyze terms that he has composed himself; the constraint solving below actually exploits that this is not the case for most reasonable intruder theories [12].

We make here an additional transformation that allows us to easily use later the result from [12] without further restricting the result of this paper. Paper [12] assumes that all functions except constants are *public*, i.e., there is an intruder generate rule for them. To model a private function f of arity n, one can then use a public function f' of arity $n+1$ where the additional argument is used for a constant that the intruder does not know (and that is not for any other purpose). For the ease of use, we assume that TASLan specifications may contain private functions, but using this simple transformation we can get rid of them, and in the following with thus – without loss of generality – assume to work with a TASLan specification where all non-constant functions are public.

2.2 Semantics

Model Relation An interpretation $\mathcal{I}$ maps from all variables to ground terms. ϕ,ψ range over all logical constructions above. We define a relation $\mathcal{I}, S \models \phi$ that says whether a pair of an interpretation $\mathcal{I}$ and a state S is a *model* of the formula ϕ:

$$\begin{array}{ll}
\mathcal{I}, S \models P & \text{iff } \mathcal{I}(P) \in \mathcal{I}(S) \\
\mathcal{I}, S \models t_1 = t_2 & \text{iff } \mathcal{I}(t_1) = \mathcal{I}(t_2) \\
\mathcal{I}, S \models \phi \wedge \psi & \text{iff } \mathcal{I}, S \models \phi \text{ and } \mathcal{I}, S \models \psi \\
\mathcal{I}, S \models \neg\phi & \text{iff } \mathcal{I}, S \not\models \phi \\
\mathcal{I}, S \models \exists X.\phi & \text{iff exists ground } t : \mathcal{I}[X \mapsto t], S \models \phi
\end{array}$$

We also say ϕ is satisfiable iff it has a model. Other constructs are defined as syntactic sugar as standard, e.g. $\forall X : \phi$ as $\neg\exists X : \neg\phi$. For a statement $\mathcal{I}, S \models \phi$ we may omit $\mathcal{I}$ if ϕ is closed (i.e. $fv(\phi) = \emptyset$), and we may omit S if ϕ does not contain predicates.

As standard, define $\phi \models \psi$ if all models of ϕ are also models of ψ; and $\phi \models\!\models \psi$ if both $\phi \models \psi$ and $\psi \models \phi$.

Least Herbrand Models For the semantics of transition rules, we need to define the least closure of a state under the Horn clauses. Let $\hat{H}$ be the conjunction of the Horn clauses of a given ASLan specification. This induces the following closure operation on states: for any ground state S, $HC(S)$ is the least set $S' \supseteq S$ such that: $P \in S'$ if $\hat{H} \wedge S' \models P$. Note that here and in the following, we treat a conjunction $S = P_1 \wedge \ldots \wedge P_n$ of predicates also as a set of predicates $S = \{P_1, \ldots, P_n\}$.

With our definition of the $\models$ relation and the least Horn closure we have chosen one interpretation of first-order terms that are often referred to as *free models* or *least Herbrand models*, which are the semantical basis for logic programming languages like Prolog. In particular, all terms are interpreted in the Herbrand universe (which is here the free algebra) and, in a given state S, all predicate symbols are interpreted by the least relations that are consistent with the Horn clauses and S. This relation is uniquely defined for Horn clauses.

Transition Relation Define $S \Rightarrow S'$ if there is a rule $\hat{L} =[\boldsymbol{X}]\Rightarrow S_R$ and interpretation $\mathcal{I}$ such that $\mathcal{I}, HC(S) \models \hat{L}$ and $\mathcal{I}(\boldsymbol{X})$ are fresh constants and $S' = S \setminus \mathcal{I}(PosE(\hat{L})) \cup \mathcal{I}(S_R)$.

Several notes are in order. The implicit consequences $HC(S) \setminus S$ of a state S are never "explicified", i.e. they are not carried over to S'. Recall that $PosE(\cdot)$ does not include the persistent predicates, so all persistent predicates of S are still contained in S'. Further, this definition does not care about type specifications. As a consequence of the ASLan conditions, all reachable states $\{S \mid S_0 \Rightarrow^* S\}$ (for initial state S_0 of the specification) are ground.

Example 2. In the specification of Table 2, the Horn closure of the initial state contains $xs(a, f1) \wedge xs(b, f2)$. If we take the first transition rule form the initial state for $A = a$, this removes the predicate $mem(a, f1)$ and thus the Horn closure of that state no longer contains $xs(a, f1)$. So in each state, the Horn closure is computed anew; all consequences that are no longer derivable simply vanish. □

A state is called an *attack state* if $S \models \mathsf{attack}$. A specification is called *secure* if it has no reachable attack state.

Security in ASLan (and even just the transition relation $S \Rightarrow S'$) is undecidable, since the Horn clauses (using untyped arguments) capture logical programming. It is still semi-decidable, because we do not allow negated implicit predicates in transition rules.

Definition 1 (Typed Model). *We say $\mathcal{I}$ is a* well-typed interpretation *if $\mathcal{I}(X) : \Gamma(X)$ for all variables X. We define a* typed model *of an ASLan specification as a variant of the above semantics where all notions are restricted to well-typed interpretations.*

In other words, our default semantics ignores all type information (because an intruder in reality is always able to send ill-typed terms) but we can choose to restrict the interpretation to well-typed terms. We show below that for all TASLan specifications it holds that, if an attack exists, then also an attack in the typed model exists. Thus in TASLan, the restriction to a typed model is sound.

3 A Symbolic Representation

We now introduce a symbolic representation of the infinite transition system that will pave the way for an effective decision procedure for the TASLan fragment when bounding the length of traces.

Symbolic States A symbolic state is generalization of a normal state, which may contain variables and constraints. We define its syntax as follows:

$\phi ::=$	Symbolic state
P	Predicate
$S \vdash P$	Deduction constraint
$\neg\exists \boldsymbol{X} : s_1 = t_1 \wedge \ldots \wedge s_n = t_n$	Negated substitution
$X = t$	Substitution
$\phi \wedge \psi$	Conjunction

We conservatively extend the model relation w.r.t. the Horn theory $\hat{H}$ of the specification (note this case does not depend on a state S):

$$\mathcal{I}, S \models S_0 \vdash P \text{ iff } \mathcal{I}(S_0) \wedge \hat{H} \models \mathcal{I}(P)$$

Thus, the constraint $S_0 \vdash P$ is true in all those interpretations in which the predicate P can be derived from the predicates in S_0 by the Horn theory $\hat{H}$. This is a generalization of the *lazy intruder* technique [15, 16, 6] where these constraints are limited to messages in the intruder knowledge.

Thus, by the relation $\mathcal{I}, S \models \phi$, symbolic states have a semantics as representing a set of ground states (and related interpretations). Usually, this set will be infinite, but is may also be finite or even empty. We say that a symbolic state is *satisfiable* if it has a model. For ASLan this satisfiability is not decidable in general (because the Horn clauses allow for logic programming).

Symbolic Transition Relation To define a transition relation, let us first make two simplifications to transition rules. Without changing the semantics of a rule, we can remove all existential quantifiers in positive facts of a transition rule, if we just ensure by renaming that it does not occur freely in the rule. Moreover we can get rid of positive equations of the form $s = t$ as follows: compute the most general unifier σ of s and t and apply σ to the entire rule as expected.

We also use the following notations. For a rule R let $\alpha(R)$ denote a renaming of all variable symbols in R with fresh variable symbols (that do not occur previously). This is necessary in the symbolic model to keep variables of different rule applications apart. Moreover for a substitution $\sigma = [X_1 \mapsto t_1, \ldots, X_n \mapsto t_n]$ where the X_i are disjoint from the variables in t_i, let $[\sigma]$ be the logical formula $X_1 = t_1 \wedge \ldots \wedge X_n = t_n$ describing σ.

We define the symbolic transition relation (with a long arrow as compared to the ground transition relation) as follows: $\phi \Longrightarrow \psi$ iff there is a transition rule R with $\alpha(R) = \hat{L} =[\boldsymbol{X}]\Rightarrow S$, and a substitution σ such that all the following conditions hold:

- σ is a most general substitution such that $\sigma(PosE(L)) \subseteq \sigma(PosE(\phi))$. (Note that in contrast to term unification, for subset unification we get finitely many most general unifiers that are pairwise incomparable.)
- Extend σ such that the variables of $\boldsymbol{X}$ (that are freshly created in the transition) are replaced by fresh constants.

– For every implicit predicate $P \in PosI(L)$ let $\chi_P = Pos(\sigma(\phi)) \vdash \sigma(P)$; denote with $\hat{\chi}$ their conjunction.
– Let Φ be the least conjunction of negated substitutions such that
  • for every negative fact $\neg\exists \boldsymbol{X} : P$ of $\sigma(\hat{L})$ and every positive fact P' of $\sigma(\phi)$, if τ is the most general unifier of P and P', then $(\neg\exists \boldsymbol{X}.[\tau]) \in \Phi$.
  • every negative equation of $\sigma(\phi)$ is also contained in Φ.
– $\psi = \sigma(\phi) \setminus \sigma(PosE(L)) \wedge \sigma(S) \wedge \Phi \wedge \hat{\chi} \wedge [\sigma]$.

Example 3. Extending our toy example from Table 2, we model that our system can process signed commands from an administrator (who would be modeled using similar rules). In this simplistic example we omit replay and eavesdropping protections:

$$admin(A, K) \wedge ik(sign(K, [add, A, B, G]_4)) \wedge A \neq B \wedge \neg mem(A, G) \\ \Rightarrow mem(B, G) \tag{1}$$

Suppose here $admin(A, K)$ expresses that A is an administrator who can issue commands with private signature key K. The command in this example is to add an agent B to group G and has the format $[add, A, B, G]_4$ where $[\cdot]_4$ represents a 4-tuple and add is a tag/command name. We discuss this way of modeling plaintext structures in Section 4.1. The rule excludes both that A can add her/himself to a group and that A can add somebody to a group he/she belongs to.

Consider now that the intruder is one of the system administrators; then he can form any kind of commands himself and send them to the service—this choice of commands is infinite. Rules with $ik(\cdot)$ on the left-hand side often give rise to an infinite ground state space, and even with typing restrictions to a very large space. In contrast, the symbolic transition system has only one successor state per rule application. Consider for instance the state:

$$\phi = admin(i, ki) \wedge mem(a, g1) \wedge mem(i, adm) \wedge \\ ik(ki) \wedge ik(a) \wedge ik(b) \wedge ik(i) \wedge ik(g1) \wedge ik(g2) \wedge ik(adm)$$

We can apply the symbolic transition relation for rule (1) under the unifier $\sigma = [A \mapsto i, K \mapsto ki]$ to match the positive explicit fact $admin(A, K)$ (in general the rule variables have to be renamed in order to avoid collisions with variables in the given state, but here we started with a ground state). From the $ik(\cdot)$ fact of the rule, we obtain the constraint $\phi \vdash ik(sign(ki, [add, a, B, G]_4))$. Note that the rule variables B and G remain uninstantiated. From the negative conditions of the rule we obtain the constraints $a \neq B \wedge G \neq adm$. The symbolic successor state consists of $\sigma(\phi)$ together with the noted constraints and the (uninstantiated) right-hand side fact $mem(B, G)$. This single symbolic state comprises all the infinitely many choices of the intruder (any messages for B and G that satisfy the constraints). This includes choices where B is not an agent name and G is not a group name, but as we later show, such ill-typed solutions are never interesting for the intruder when the specification satisfies the type-unambiguity rules of TASLan. □

The following lemma shows that the symbolic transition system is a correct representation of the ground transition system:

Lemma 1. *Let* $[\![\phi]\!] = \{S \mid \exists \mathcal{I} : \mathcal{I}, S \models \phi\}$. *Then for all symbolic states* ϕ*:*

$$\{S' \mid \exists \psi : \phi \Longrightarrow \psi \wedge S' \in [\![\psi]\!]\} = \{S' \mid \exists S : S \in [\![\phi]\!] \wedge S \Rightarrow S'\} \; .$$

As a consequence, a satisfiable symbolic state that contains the predicate attack *is reachable using* $\Longrightarrow$ *from initial state* S_0 *in* l *steps iff a ground attack state is reachable using* $\Rightarrow$ *from* S_0 *in* l *steps.*

Proof. We consider a symbolic state ϕ and a transition rule $R = \hat{L} =[\boldsymbol{X}]\Rightarrow S_R$ and show that the symbolic transition relation mirrors what the ground transition relation does for the states $[\![\phi]\!]$.

- First let us consider the positive explicit predicates of $\hat{L}$. By definition, they cannot be produced by the Horn clauses but only by transition rules. Thus, for R to be applicable, it must be possible to match them with the positive predicates $PosE(S)$ of a ground state $S \in [\![\phi]\!]$. On the symbolic level this corresponds to computing the most general subset unifiers σ between $PosE(\phi)$ and $\hat{L}$. From here on, all further steps are under one such σ, restricting the interpretations of ϕ to those that are instances of σ (w.r.t. $fv(\phi)$).
- Second let us consider the other positive predicates $S_I = PosI(\hat{L})$—the implicit predicates. In the ground case $S \in [\![\phi]\!]$ the transition relation requires that the $\mathcal{I}(S_I)$ are contained in $HC(S)$. On the symbolic level that means that we require $Pos(\sigma(\phi)) \vdash \sigma(P)$ for any predicate of S_I. So here we postpone the computation of the instances that satisfy the conditions by putting constraints.
- Third we have the negative predicates. Again by the definition of ASLan, the negative predicates must be explicit predicates, i.e., they do not depend on the Horn closure. In the ground case $S \in [\![\phi]\!]$ we thus have to check that no negative predicate matches with a predicate in S. Correspondingly, on the symbolic level we must exclude all interpretations $\mathcal{I}$ under which a negative predicate $\neg\exists \hat{X} : \sigma(P)$ is interpreted the same as a positive predicate $\sigma(P')$ of $\sigma(\phi)$ (for suitable $\hat{X}$). Thus we compute the most general unifier τ of $\sigma(P)$ and $\sigma(P')$ and exclude all instances of τ in ψ (existentially quantifying the variables of $\hat{X}$).
- Fourth, for the negative equations: we just state them in the ψ, so we exclude all interpretations of $\sigma(\phi)$ that do not satisfy these equations.
- Under all interpretations that survived the restrictions, the transition is possible, removing the predicates $\sigma(PosE(\hat{L}))$ and introducing $\sigma(S)$; in both symbolic and ground case, we have $\hat{X}$ of course replaced with fresh constants. □

We now distinguish several kinds of constraints in a symbolic state and we tackle each of them in isolation and before we look at their interaction:

- Intruder deduction constraints $S \vdash P$ where P and all predicates in S are of the form $\mathsf{ik}(t)$ for some term t.

– Other deduction constraints $S \vdash P$ where no predicate is of the form $\mathsf{ik}(t)$.
– Negated substitutions $\neg\exists \boldsymbol{X} : s_1 = t_1 \wedge \ldots \wedge s_n = t_n$.
– Substitutions $X = t$. Our constructions will ensure that the variable X does not occur elsewhere, and this kind of (always satisfiable constraint) is just to remember partial solutions, i.e. all models of the containing symbolic state must satisfy $\mathcal{I}(X) = \mathcal{I}(t)$.

The satisfiability of negative equalities is straightforward to check: for $L = \neg\exists \boldsymbol{X} : s_1 = t_1 \wedge \ldots \wedge s_n = t_n$ check the unification problem $\tau((s_1, t_1), \ldots, (s_n, t_n))$ for a substitution τ that replaces all free variables L (i.e. those that are not quantified in $\boldsymbol{X}$) with fresh constants (of the appropriate type). There is a unifier iff L is unsatisfiable.

We show below that satisfiability of intruder deduction constraints is also decidable, slightly extending known results. However, satisfiability of other constraints is not decidable for ASLan in general, since we can use Horn clauses for logic programming.

4 Type Ambiguity-Free Specifications

We now introduce a fragment of ASLan, called TASLan: basically the format of messages (and predicates) must be different whenever their intended type is different. We show that security is decidable for TASLan if the length of traces is bounded; more precisely, this problem is NEXPTIME complete. Note that the restriction in TASLan is not a typed model directly, but rather a generalization of the tagging principle; however we do not prescribe a particular way of disambiguating messages. We show—as a side result of our decision procedure for bounded-length TASLan—that a typed model is sound (even without any bounds on the length of traces).

We proceed as follows. We first introduce the fragment TASLan, and then show that for symbolic states in TASLan, we can decide the satisfiability of all constraints. This gives an effective procedure for bounded-length traces. Finally we give the typing result (that the typed model is "relatively sound" for TASLan), and show how this can be used for different kinds of automatic verification methods other than our symbolic method.

Definition 2 (SMP, adaption from [12]). *The set of sub-message patterns, $SMP(M)$, of a set of terms M is the least set closed under the following rules:*

1. *If $t \in M$ then $t \in SMP(M)$*
2. *If $t \in SMP(M)$ and t' is a subterm of t then $t' \in SMP(M)$*
3. *If $t \in SMP(M)$ and δ is a well-typed substitution then $\delta(t) \in SMP(M)$*
4. *If $f(\boldsymbol{t}) \in SMP(M)$ and there is an intruder analysis rule*

$$\forall \boldsymbol{X}.\mathsf{ik}(f(\boldsymbol{X})) \wedge \mathsf{ik}(s_1) \ldots \wedge \mathsf{ik}(s_k) \implies \mathsf{ik}(X)$$

then $\delta(s_1), \ldots, \delta(s_n) \in SMP(M)$ for $\delta = \boldsymbol{X} \mapsto \boldsymbol{t}$.

This definition captures all those terms that occur in intruder constraints, if M is the set of messages that occur in the protocol specification. We use it to define a condition similar to type-flaw resistance in [12]:

Definition 3 (Adaption type-flaw resistance in [12]). *TASLan is the fragment of ASLan specifications with the following additional requirements/modifications:*

- *Every predicate except* ik *has a type in which* untyped *does not occur.*
- *For every predicate* $\mathsf{ik}(t)$ *in the transition rules,* t *has a type in which* untyped *does not occur.*
- *Let* M *be the set of messages* t *such that* $\mathsf{ik}(t)$ *occurs in a transition rule. Then for every* $t, t' \in SMP(M)$ *that are not variables it holds that* $\Gamma(t) = \Gamma(t')$ *if* t *and* t' *are unifiable.*
- *All bound variables have an atomic type and all variables in inequalities and predicates except* ik *have atomic types.*
- *For every equation* $s = t$, $\Gamma(s) = \Gamma(t)$.

[2]

We also assume that the intruder can always generate fresh elements of any type in any state, so that for instance the constraint $\mathsf{ik}(X) \wedge \mathsf{ik}(Y) \wedge X \neq Y$ is always satisfiable. While it is natural to "grant" this to the intruder, it is tricky to formulate this, because we actually need transition rules to freshly generate new intruder constants. We silently assume such rules, and note that our lazy treatment of constraints below gives this property for free: a constraint like the above is simply considered as a solved form (without making actual transitions for creating two concrete values for X and Y).

[2] The previous definition was more permissive:

Definition 4. *TASLan is the fragment of ASLan specifications with the following additional requirements/modifications:*

- *Every predicate except* ik *has a type in which* untyped *does not occur.*
- *For every predicate* $\mathsf{ik}(t)$ *in the transition rules,* t *is non-atomic and has a type in which* untyped *does not occur.*
- *Let SMP be the non-atomic subterms of all terms* t *that occur in a predicate* $\mathsf{ik}(t)$ *in the transition rules,* α*-renamed so that two distinct elements of SMP have no variables in common. Whenever there is a unifier for two* $t_1, t_2 \in SMP$*, then* t_1 *and* t_2 *must have the same type.*

The original definition is too permissive: inequalities with variables of composed types may have only ill-typed solutions, but no well-typed ones. This means we have to forbid inequalities with composed-type variables. This means also a corresponding restriction on predicates, since negative checks for a predicate can in the constraint reduction introduce corresponding inequalities. We have here opted for a restriction that is guaranteed to satisfy the requirements for applying the result from [12], and we are investigating where we can relax these conditions.

4.1 How Restrictive is TASLan?

As indicated in the add command in Example 3 (which of course falls into the TASLan fragment), we model concatenation by the family $[\cdot]_n$ of n-tuple operators (for $n > 1$). This model abstracts from several implementation details, such as field lengths or special tags that mark the beginning and end of fields—we simply assume that the implementation has a unique way to decompose every acceptable message into its components. This is a reasonable requirement to the implementation that excludes many low-level attacks. Tags like *add* in the example then are an easy way to disambiguate messages. (Alternatively, one can instead introduce new functions, e.g. $add(A, B, G)$ in example, and give the intruder rules for composing/decomposing them.)

Basically, we thus see every kind of plaintext message like a *paper form* that has a well-defined set of fields. Many ASLan specifications are already written in this style—independent of our work. With this "form approach", almost all specifications meet the requirements of TASLan. This is because we exclude with a single tag any confusions between different forms that carry similar information but with different meaning.

Many ASLan specifications, and even more protocols, do not use this regime and thus do not immediately fall into the TASLan fragment. To use the most cited example, the encrypted content of the first two messages of NSPK—the pairs NA, A and NA, NB—already violate our requirements because NA and NB are random numbers while A is an agent name. (In fact, this ambiguity gives rise to a type-flaw attack [14].) Our approach would be to identify the ambiguities and resolve them; the messages may then be $[nspk1, NA, A]_3$ and $[nspk2, NA, NB]_3$ for instance, and this variant falls into the TASLan fragment.

We propose that in this way every protocol can be transformed into a reasonable TASLan model, but in doing so one may exclude some potential low-level type-flaw or parsing attacks. However the transformation process gives clear indications where problems could arise and what we require from the implementation. Thus one could say that TASLan requires, and exploits, what good engineering practice demands in the first place.

4.2 Symbolic Horn Closure

Let $\hat{H}$ be the conjunction of Horn clauses without intruder deduction (which we handle separately). We want to consider the Horn closure under $\hat{H}$ for symbolic states. In general, this closure is infinite in ASLan (due to instantiation of variables), but we will show it is finite in TASLan. For that, we define the following evaluation relation over symbolic states:

Definition 5. *Let $\hat{H}$ be the conjunction of Horn clauses without intruder deduction. $\phi \hookrightarrow \psi_1 \vee \psi_2$ if there is a Horn clause $H_R \in \hat{H}$ such that*

- *$\alpha(H_R) = \forall \hat{X} : S \to P$ for a renaming α of variables in H_R,*
- *S unifies with a subset of $Pos(\phi)$ under the most general unifier σ,*
- *$\psi_1 = \sigma(\phi \wedge P)$ and $\psi_2 = \neg[\sigma]$,*

– $\sigma(P) \notin \phi$ *(so the predicate is indeed newly derived)*
– *The negative equation constraints in* ψ_1 *are satisfiable.*

We extend $\hookrightarrow$ *to a relation on disjunctions of symbolic states as expected. We say* $\phi_1 \vee \ldots \vee \phi_n$ *is a* normal form *(for Horn theory* $\hat{H}$*) if it has no successor modulo* $\hookrightarrow$*.*

The $\hookrightarrow$ can be understood as follows: at every reduction step we check whether a new predicate (that is not yet present in ϕ) is derivable in one step under a substitution σ. Note that we are not forced to take the substitution σ, because this only represents a subset of the ground states represented by ϕ in which the new predicate $\sigma(S)$ is derivable. All the other states are represented by $\neg[\sigma]$ (and in those, $\sigma(S)$ is in general not derivable). Thus each $\hookrightarrow$ step makes a case split into states that satisfy σ and those that do not. In order to have a notion of normal form without enforcing any substitution σ, we have the condition that requires that the negative equalities in ϕ_1 are satisfiable: if we have entered a case with $\neg[\sigma]$, then we cannot actually apply σ to that symbolic state anymore.

Example 4. Consider the Horn clauses from Table 2 and the following symbolic state (which can occur in a specification with more transition rules):

$$\phi = \mathit{mem}(a, \mathit{g1}) \wedge \mathit{own}(\mathit{g1}, \mathit{f1}) \wedge \mathit{mem}(\mathit{A2}, \mathit{G2}) \wedge \mathit{own}(\mathit{g2}, \mathit{f2}) \wedge \mathit{deputy}(a, \mathit{A3})$$

Note that here for instance $\mathit{G2}$ is a variable, and $\mathit{g2}$ a constant. One possible derivation with $\hookrightarrow$ is as follows:

$$\begin{aligned}
\phi &\hookrightarrow \underbrace{(\phi \wedge \mathit{xs}(a, \mathit{f1}))}_{\phi_1} \vee (\phi \wedge \mathit{false}) \\
\phi_1 &\hookrightarrow \underbrace{(\phi_1[\mathit{G2} \mapsto \mathit{g1}] \wedge \mathit{xs}(\mathit{A2}, \mathit{f1}))}_{\phi_2} \vee \underbrace{(\phi_1 \wedge \mathit{G2} \neq \mathit{g1})}_{\phi_3} \\
\phi_3 &\hookrightarrow \underbrace{(\phi_3[\mathit{G2} \mapsto \mathit{g2}] \wedge \mathit{xs}(\mathit{A2}, \mathit{f2}))}_{\phi_4} \vee \underbrace{(\phi_3 \wedge \mathit{G2} \neq \mathit{g2})}_{\phi_5} \\
\phi_4 &\hookrightarrow \underbrace{(\phi_4[\mathit{A2} \mapsto \mathit{A3}] \wedge \mathit{xs}(a, \mathit{f2})) \vee (\phi_4 \wedge \mathit{A2} \neq \mathit{A3})}_{\phi_6}
\end{aligned}$$

We thus have $\phi \hookrightarrow^* \phi_2 \vee \phi_5 \vee \phi_6$ which is a normal form—for instance if we try in ϕ_2 to apply the second Horn clause (under $\mathit{A2} = a$ or under $\mathit{A2} = \mathit{A3}$) we get only the already present fact $\mathit{xs}(a, \mathit{f1})$. □

Lemma 2. $\hookrightarrow$ *is convergent modulo* $\models\!\!\!\models$ *for TASLan, while for ASLan in general it is not terminating (but confluent).*

Proof. For $\phi \hookrightarrow \psi$ follows $\phi \models\!\!\!\models \psi$ (as we make a case split into interpretations that support a substitution σ and those interpretations that do not). From this follows immediately confluence modulo $\models\!\!\!\models$.

For ASLan specifications, $\hookrightarrow$ does not terminate in general: e.g. consider the Horn clause $\forall X : p(X) \to p(f(X))$ for untyped X, f, and p.

For TASLan specifications, thanks to the typing, we get termination as follows. In all predicates of the symbolic state ϕ (except ik which we do not consider in $\hookrightarrow$) we can annotate the variables with their intended type (and that cannot contain $\mathsf{untyped}$). No unification of the predicates of a Horn clause with positive predicates of ϕ can destroy well-typedness here, so all derived clauses in $\hookrightarrow$ are still well-typed. The unification can introduce new variables, because we unify predicates of the symbolic state with the premises of the clause. Since we α-rename all variables in the clause with fresh variables before this unification step, we ensure that clauses and states always have disjoint sets of variables. Consider any variable $X : \tau$ of a symbolic state; in a unification we can distinguish three cases:

- X is replaced with another variable $Y : \tau$ of the state.
- X is replaced with another variable $Y : \tau$ of the α-renamed Horn clause—this is logically equivalent to conversely replacing Y by X.
- X is unified with a more concrete term.

If we handle the second case by renaming Y to X instead (note that we are computing Horn closure modulo $\models\!\!|$), the only case that can actually introduce new variables into a symbolic state is thus the third case. These new variables can only be of the appropriate subtype. We can therefore derive a bound on the set of all variables, constants, and composed terms for any type that occurs in a conclusion of a Horn clause. Thus we have an upper bound for the derivable symbolic predicates, bounding also the possible derivations with $\hookrightarrow$. □

Combining the previous results, we get:

Lemma 3. *Satisfiability of symbolic states ϕ of TASLan without considering intruder deduction constraints is decidable.*

Proof. Pick a constraint of the form $S \vdash P$ (except intruder deduction) from ϕ. Compute the symbolic closure (using $\hookrightarrow$) of S under the Horn theory $\hat{H} \cup \{P \to \mathsf{sat}\}$ for a fresh symbol sat (where $\hat{H}$ are the Horn clauses of the specification, without intruder deduction). This yields a disjunction $\psi_1 \vee \ldots \vee \psi_n$. It is immediate that ψ_i contains the predicate $\mathsf{sat}()$ if the deduction constraints are satisfied in all instances represented by ψ_i. Moreover, in those ψ_i that do not contain $\mathsf{sat}()$, *no* instantiation of ψ_i satisfies the deduction constraints.

Let σ_i be the substitutions performed to derive ψ_i. Let finally ϕ' be ϕ without the constraint $S \vdash P$ we had picked. We have:

$$\phi \models\!\!| \bigvee_{i|\mathsf{sat}\in\psi_i} \sigma_i(\phi') \wedge [\sigma_i]$$

i.e. we can eliminate any $S \vdash P$ constraint by applying the those substitutions σ_i to ϕ' under which P is actually derivable from P.

With this procedure we can step by step eliminate all $S \vdash P$ constraints except intruder deduction. It remains to check the inequalities for satisfiability (which we can do also at any intermediate step). □

The proof in fact gives us a procedure to obtain from ϕ an equivalent disjunction $\psi_1 \vee \ldots \vee \psi_n$ of symbolic states where all $S \vdash P$ constraints (except intruder deduction) are eliminated and the remaining inequalities constraints are all satisfiable.

4.3 Lazy Intruder Constraint Reduction

We now turn to checking the satisfiability of intruder constraints of the form $S \vdash P$ where all predicates of S and P are of the form $\mathsf{ik}(t)$. An important property for the lazy intruder deduction is that they are well-formed:

Definition 6. *A conjunction of intruder deduction constraints is called* well-formed *if we can order them as $S_1 \vdash P_1 \wedge \ldots \wedge S_n \vdash P_n$ such that*

- *$S_{i+1} \implies S_i$ for $0 \leq i < n$, i.e. the intruder knowledge grows monotonically.*
- *$fv(S_i) \subseteq \bigcup_{0 \leq j < i} fv(P_j)$, i.e. all variables in the constraints first occur from a message the intruder generated.*

We call an intruder constraint $S \vdash \mathsf{ik}(t)$ simple if t is a variable. A simple constraint is always satisfiable (because the intruder can generate fresh terms of any type as discussed before).

In a symbolic state that is reachable from a ground initial state, we order the constraints in the order they have been created. The intruder knowledge grows monotonically because $\mathsf{ik}(\cdot)$ is persistent. The condition on variable occurrence however does not hold for reachable symbolic states in general: variables may as well be "introduced" by other (non-intruder) constraints of the form $S \vdash P$. However, after performing the symbolic Horn closure, these constraints are all gone, and the respective variables can be substituted by terms that can only contain variables that occur elsewhere in the state—i.e. introduced by intruder constraints.

Theorem 1 (Adaption of [16]). *Satisfiability of well-formed intruder deduction constraints is NP-complete. Moreover, there is a procedure that transforms a well-formed ϕ into a finite disjunction of well-formed intruder deduction constraints $\psi_1 \vee \ldots \vee \psi_n \models\!\!\models \phi$ ($n \geq 0$) such that every ψ_i is simple.*

Proof. This proof follows the standard lazy intruder technique [15, 16, 6].

We first give a set of rules $\dfrac{\psi}{\phi}$ that can be read as "if ψ is satisfiable then also ψ". They mirror the ground intruder deduction rules on the symbolic level: the intruder can use terms from his knowledge to satisfy a constraint, he can compose terms from his knowledge (respectively: decompose the terms to generate), and he can analyze terms in his knowledge, provided he can derive the necessary keys

from his knowledge:

$$\frac{\sigma(\phi) \wedge [\sigma]}{(\mathsf{ik}(t) \wedge S \to \mathsf{ik}(s)) \wedge \phi}\ (\mathit{Unify})\quad t, s \notin \mathcal{V}, \sigma = mgu(s,t)$$

$$\frac{(S \to t_1 \wedge \ldots \wedge S \to t_n)}{(S \to \mathsf{ik}(f(t_1,\ldots,t_n))) \wedge \phi}\ \begin{pmatrix}\forall \boldsymbol{X} : \mathsf{ik}(X_1) \wedge \ldots \wedge \mathsf{ik}(X_n) \\ \to \mathsf{ik}(f(X_1,\ldots,X_n))\end{pmatrix} \in \hat{H}$$

$$\frac{\begin{array}{l}(\mathsf{ik}(f(\boldsymbol{s})) \wedge \mathsf{ik}(\sigma(X)) \wedge S \vdash \mathsf{ik}(u)) \\ \wedge (S \vdash \mathsf{ik}(\sigma(t_1))) \wedge \ldots \wedge (S \vdash \mathsf{ik}(\sigma(t_n))) \wedge \phi\end{array}}{(\mathsf{ik}(f(\boldsymbol{s})) \wedge S \vdash \mathsf{ik}(u)) \wedge \phi}\ \begin{pmatrix}\forall \boldsymbol{X} : \mathsf{ik}(f(\boldsymbol{X})) \wedge \mathsf{ik}(t_1) \wedge \ldots \\ \wedge \mathsf{ik}(t_n) \to \mathsf{ik}(X)\end{pmatrix} \in \hat{H}, \sigma = \boldsymbol{X} \mapsto \boldsymbol{s}$$

[3]

The proof of soundness, completeness and termination has been conducted in Isabelle/HOL [12] with a few notational differences. We keep the original proof sketch as the outline of the proof is still correct.

Soundness is straightforward: For each rule $\frac{\psi}{\phi}$, from a satisfying interpretation of an instance $\tau(\psi)$ of ψ, we can derive an interpretation that satisfies $\tau\phi$.

Completeness: Consider a satisfiable non-simple constraint ϕ, and a satisfying interpretation $\mathcal{I}$. Thus for every $S \vdash P$ in ϕ we can label P with a proof tree that $\mathcal{I}(P)$ can be derived from $\mathcal{I}(S)$ using the intruder deduction Horn clauses. Show that there is at least one constraint reduction step that supports $\mathcal{I}$ and that mirrors a deduction step that some non-simple P is labeled with. (Here we need to exploit well-formedness, because we need to exclude that a predicate $\mathsf{ik}(X)$ in the intruder knowledge ever needs to be analyzed.)

Termination: The way we formulate it here, analysis steps may lead us into a non-termination, if repeatedly analyzing the same term. If we exclude that, our proof relationship can produce only finitely many irreducible constraints. All irreducible constraints are either unsatisfiable or simple (because non-simple, satisfiable constraints can be further reduced due to completeness).

The length of derivations is polynomial in the size of the constraints. Note that substitutions may give an exponential blow up in ASLan in general (e.g.

[3] The third rule was originally:

$$\frac{\begin{array}{l}(\mathsf{ik}(t) \wedge \mathsf{ik}(s) \wedge S \vdash \mathsf{ik}(u)) \\ \wedge (S \vdash \mathsf{ik}(t_1)) \wedge \ldots \wedge (S \vdash \mathsf{ik}(t_n)) \wedge \phi\end{array}}{(\mathsf{ik}(t) \wedge S \vdash \mathsf{ik}(u)) \wedge \phi}\ \begin{pmatrix}\forall \boldsymbol{X} : \mathsf{ik}(t) \wedge \mathsf{ik}(t_1) \wedge \ldots \\ \wedge \mathsf{ik}(t_n) \to \mathsf{ik}(s)\end{pmatrix} \in \hat{H}, t \notin \mathcal{V}$$

The problem is here that we had written both t for the term being analyzed in the analysis rule and in the constraint. In fact these may be different terms with variables, so actually this rule would require first a unification between rule-term and constraint-term. (This is now indeed in the new rule the substitution σ.) This could actually lead to problems if the analysis could induce any substitution of constraint-level variables; thus we have reduced it to the analysis rules allowed in [12]. The soundness, completeness, and termination has in this case actually been proved in Isabelle/HOL.

$[X_0 \mapsto f(X_1, X_1)]$, followed by $[X_1 \mapsto f(X_2, X_2)]$ and so on) but exponential running time here can be avoided when representing substitutions as a DAG [16]. We can thus derive a polynomial time non-deterministic machine that accepts a constraint iff it is satisfiable (in each step "non-deterministically guessing" which intruder rule to apply next). Thus the satisfiability is in NP.

NP-Hardness (even for TASLan):[4] Given a Boolean formula ϕ in CNF with variables $(X_1, \ldots, X_n)$. We construct an intruder deduction problem from ϕ so that the intruder can first choose true or false for each X_i:

Define $M_0 = \{h(1, c_0), h(1, c_1), h(2, c_0), h(2, c_1), \ldots, h(n, c_0), h(n, c_1)\}$ where constants c_0 and c_1 are not known to intruder, and

$$tr_C(\phi) = \bigwedge_{i=1}^{n} \{\mathsf{ik}(m) \mid m \in M_0\} \vdash \mathsf{ik}(h(i, X_i))$$

We further have a constraint with a knowledge M that includes besides M_0 the following terms that depend on the intruder's previous choices X_i:

- $f(i, X_i)$ for $1 \leq i \leq n$
- $scrypt(f(i, c_0), s_k)$ if clause number k of ϕ contains $\neg X_i$ as a literal
- $scrypt(f(i, c_1), s_k)$ if clause number k of ϕ contains X_i as a literal
- $scrypt(g(s_0, \ldots, s_l), s)$ for l the number of clauses in ϕ.

Thus for each choice X_i the intruder now knows the term $f(i, X_i)$. Here *scrypt* stands for symmetric encryption and we have the decryption rule

$$\mathsf{ik}(scrypt(X, Y)) \wedge \mathsf{ik}(X) \rightarrow \mathsf{ik}(Y) \ .$$

Then he can derive the secret s_k iff the kth clause contains a literal that becomes true under the his choice of the X_i. If he knows all s_k, i.e., when the entire formula is true under his choice, he obtains the main secret s. Thus the constraint

$$tr(\phi) = tr_C(\phi) \wedge \{\mathsf{ik}(m) \mid minM\} \vdash \mathsf{ik}(s)$$

is satisfiable iff ϕ is. □

Together we now have:

Lemma 4. *Satisfiability is decidable for reachable symbolic states of TASLan specifications, and thus whether an attack state is reachable in l steps or less.*

Proof. Let ϕ be a reachable symbolic state.

- First we solve all $S \vdash P$ constraints of ϕ that are not intruder deduction constraints according to Lemma 3 and obtain symbolic states $\psi_1 \vee \ldots \vee \psi_n$. These must all be well-formed since the state is reachable (i.e. variables can only be introduced by the deduction constraints, and the ψ_i can contain only intruder deduction constraints).

[4] The proof assumes that the set of symbols and intruder deduction rules is not fixed; when this is fixed, we can still give a similar proof for ASLan, but for TASLan, we have then a fixed number of satisfiable $S \vdash P$ constraints (modulo variable renaming) so the decision problem is then trivially in linear time.

– We next apply Theorem 1 to each ψ_i to solve the intruder constraints and obtain a new disjunction $\chi_1 \vee \ldots \chi_m$ in which all intruder deduction constraints are simple.
– We check the negative substitutions in the χ_i. (Note that some substitutions performed in the lazy intruder constraint reduction may render the negative substitution constraints unsatisfiable, then the respective χ_i is ruled out.)

We thus have at this point a set of satisfiable simple intruder deduction constraints, i.e. of the form $M \vdash \mathsf{ik}(X)$, and a conjunction of negative substitution constraints $\neg\exists \boldsymbol{X}.s_1 = t_1 \wedge \ldots \wedge s_n = t_n$. Moreover all free variables of the negative substitutions occur as variables of the $M \vdash \mathsf{ik}(X)$. The question is whether this conjunction of intruder constraints and inequalities is satisfiable.

Recall that the procedure to check the satisfiability of negative substitutions is based on replacing all free variables with fresh constants of the respective types (and then trying to unify the resulting $s = t$ which is possible iff the negative substitution is unsatisfiable). Even if the intruder knowledge in any of the constraints does not contain enough different constants of the respective types, in an untyped model the intruder can simply generate different terms for every free variable X and then the negative substitutions are also satisfied.

Thus, the inequalities may force us into an ill-typed substitution of variables when the intruder knowledge does not have enough "diversity" of constants. For this reason we had assumed in the definition of TASLan that we have intruder rules for creating fresh constants of any type. Thus, if we have an attack where the inequalities require ill-typed substitutions of variables, there is also an attack where the intruder first created sufficiently many fresh constants of the respective types so that all constraints can be solved in a well-typed form.

Note that our symbolic model gives us the fresh constants "for free": we simply stop when we have reached simple constraints and satisfiable inequalities since we know this is satisfiable, even in a typed model if we had generated enough fresh constants. □

4.4 Organizing Search

With this, we have generalized the symbolic, constraint-based decision procedures for bounded-length verification—the lazy intruder—to support Horn clauses. There are now several choices how to coordinate the different aspects of constraint reduction. When solving the constraints of a symbolic state ϕ, we usually get into a finite case split $\psi_1 \vee \ldots \vee \psi_n$ of symbolic states where each ψ_i has only constraints in a solved form. If $n = 0$ we know that ϕ is unsatisfiable and can be discarded from the search. When constructing the successor states of ϕ we can either continue with ϕ or compute the successor states of each of the ψ_i. It is in general unclear which is preferable: continuing on ϕ requires that we repeat a lot of constraint reduction work in the successor states, while continuing on ψ_i can mean a large case split into similar cases. Our current prototype is based on the ψ_i expansion, but we see room for optimization in finding a middle ground between the two extremes: sometimes being more lazy and leaving some choices open once we have established that there exists at least one solution.

4.5 Typed Model for TASLan

It is crucial that all results so far do *not* require the restriction to a typed model (Definition 1), but merely exploit the fact that TASLan requires distinct formats for messages of distinct types (Definition 3). We now use these results, in particular Theorem 1, to show that the restriction to a typed model comes without loss of attacks for TASLan specifications:

Lemma 5. *If there is an attack against a TASLan specification, then there is an attack in the typed model, i.e. where every variable of transition and Horn rules is instantiated with a term of the desired type.*

Proof. We can now use the typing result from [12] for this. Let ϕ be a reachable symbolic state. First we solve all $S \vdash P$ constraints of ϕ that are not intruder deduction constraints according to Lemma 3 and obtain symbolic states $\psi_1 \vee \ldots \vee \psi_n$. Each of these ψ_i consists only of well-formed intruder constraints plus inequalities that satisfy the type-flaw resistance criteria of [12]. Then theorem 3 of [12] guarantees a well-typed solution. □
[5]

Theorem 2 (Completely typed model is no restriction). *Every TASLan specification $\mathcal{S}$ can effectively be transformed into a specification $\mathcal{S}'$ such that*

- *In $\mathcal{S}'$ also the variables in intruder rules are typed.*
- *$\mathcal{S}$ has an attack iff $\mathcal{S}'$ has an attack (and the same holds when bounding traces to length l or less).*
- *$|\mathcal{S}'|$ is polynomial in the size of $\mathcal{S}$.*

[5] Old version of the proof:

Proof. There are only three points where unification occurs (and thus instantiation of variables):

- In checking negated substitutions.
- In computing the symbolic Horn closure.
- In the lazy intruder *Unify* rule.

As already shown above, the first two can never lead to ill-typed substitutions in our satisfiability checks. We now show that this also holds for the *Unify* rule of the lazy intruder: this rule has the property that it only unifies terms s and t that are not variables. We had required for TASLan that any non-variable subterms s and t of messages that can be sent and received (i.e. that can ever become part of a lazy intruder constraint) are unifiable iff they have the same type.

Note that this holds for any attack, independent of bounds on the length of derivations. □

The problem of this original proof: it is too sketchy for the handling of the intruder constraints (and the proof of constraint reduction it refers to was also flawed); moreover it must be related to the inequalities, as done in the proof of [12].

Proof. Instantiate all intruder rules with types that can ever occur when honest agents are sending and receiving. □

There is another way to see this: since every variable now has a completely determined type, we can turn this into a problem without function symbols: consider a predicate $p(t)$ for $t : f(\tau)$, then we could replace this with a predicate $p_f(t')$ for $t' : \tau$. This is because even if t is a variable, the typed model dictates it can only be instantiated with a term of the form $f(t')$ for $t' : \tau$, i.e. we can equivalently replace t with $f(x)$ where $x : \tau$ is a new variable. Applying this to the whole specification, we obtain a specification without function symbols. However note that we hereby replace $\mathsf{ik}(\cdot)$ with a family of predicates that represent intruder knowledge of certain functions—and they must be treated accordingly as persistent predicates that are allowed on the right-hand side of both Horn clauses and transition rules. This reflects that with the typed model we essentially turn the logic programming problem of the Horn clauses into a Datalog problem [5].

We now prove NEXPTIME completeness of TASLan insecurity when we are given a bounded length of traces ("bounded number of sessions" in the security protocol parlance):

Theorem 3. *The following problem is NEXPTIME-complete: Given a bound $l \in \mathbb{N}$, and a TASLan specification S, is an attack state reachable in l state transitions or less? Here the problem size N is the length in bits of the description of S and l together (thus $l \leq 2^N$).*

Proof. We use here the typed model of TASLan according to Theorem 2.

Bounded TASLan insecurity is in NEXPTIME. We can first derive some upper bounds for sizes of things:

- Since we require that the bound l is part of the encoding, we have $l \leq 2^N$ for the size in bits N.
- The set Σ_0 of symbols (variables, constants, functions, predicates) that can occur in an ASLan specification is also linear: $|\Sigma_0| \leq N$. Note that, since we have allow declaration of symbols, its encoding/denotation in the specification is not constant size, but rather $\log_2 N$ bits.
- The set Σ of fresh constant symbols that can occur in reachable states: $\Sigma \in O(l \cdot N)$ (because in every step, we can generate at most N new symbols).
- The universe U of all predicates that can ever be true in any ground state is therefore $|U| \in O(2^{poly(N)})$ (for $poly(N)$ some polynomial of N).
- The space of reachable ground states is thus in $O(2^{|U|})$ (which is over-exponential).
- Horn closure $HC(S)$ of a ground state S can be at most of size $|U|$ (because we can only positively derive predicates). Note this includes also intruder deduction.

Let now a non-deterministic machine "guess" a symbolic trace $S_0 \to S_1 \to \ldots \to S_k$ of length $k \leq l$: in each state S_i, choose non-deterministically a rule to apply;

Initial state:

$$length(\boldsymbol{N}) \wedge tile(t_1, N_{T_1}, S_{T_1}, E_{T_1}, W_{T_1}) \wedge \ldots \wedge tile(t_n, N_{T_n}, S_{T_n}, E_{T_n}, W_{T_n}) \wedge$$
$$f(\boldsymbol{1}, \boldsymbol{1}, t_1) \wedge \ldots \wedge f(\boldsymbol{1}, \boldsymbol{n}, t_n)$$
$$\wedge correct(\boldsymbol{1}, \boldsymbol{1}) \wedge isBool(0) \wedge isBool(1)$$

The transition rules:

$$f(\boldsymbol{I}, \boldsymbol{J}, T) \wedge succ(\boldsymbol{SJ}, \boldsymbol{J}) \wedge \neg\exists T_1 : f(\boldsymbol{I}, \boldsymbol{SJ}, T_1).tile(T_2, N, S, E, W) \Rightarrow f(\boldsymbol{I}, \boldsymbol{SJ}, T_2)$$
$$f(\boldsymbol{I}, \boldsymbol{J}, T) \wedge succ(\boldsymbol{SI}, \boldsymbol{I}) \wedge \neg\exists T_1 : f(\boldsymbol{SI}, \boldsymbol{J}, T_1).tile(T_2, N, S, E, W) \Rightarrow f(\boldsymbol{SI}, \boldsymbol{J}, T_2)$$

The Horn theory (where we omit the universal quantifiers and the predicate $isBool(X_i)$ on the left-hand sides for all Boolean variables):

$$\rightarrow succ(X_{k-1}, \ldots, X_1, 1, \quad X_{k-1}, \ldots, X_1, 0)$$

for each $2 \leq i \leq k$ the rule :
$$\rightarrow succ(X_{k-1}, \ldots, X_i, 0, 1, \ldots, 1, \quad X_{k-1}, \ldots, X_i, 1, 0, \ldots, 0)$$

$$succ(\boldsymbol{SI}, \boldsymbol{I}) \wedge succ(\boldsymbol{SJ}, \boldsymbol{J}) \wedge correct(\boldsymbol{SI}, \boldsymbol{J}) \wedge correct(\boldsymbol{I}, \boldsymbol{SJ})$$
$$\wedge f(\boldsymbol{I}, \boldsymbol{SJ}, T_1) \wedge f(\boldsymbol{SI}, \boldsymbol{J}, T_2) \wedge f(\boldsymbol{SI}, \boldsymbol{SJ}, T_3)$$
$$tile(T_1, N_1, S_1, E_1, W_1) \wedge tile(T_2, N_2, S_2, E_2, W_2) \wedge tile(T_3, S_1, S_3, E_3, E_2)$$
$$\rightarrow correct(\boldsymbol{SI}, \boldsymbol{SJ})$$

$$succ(\boldsymbol{SJ}, \boldsymbol{J}) \wedge correct(1, \boldsymbol{J}) \wedge f(1, \boldsymbol{J}, T_1) \wedge f(1, \boldsymbol{SJ}, T_2)$$
$$\wedge tile(T_1, N_1, S_1, W_1, E_1) \wedge tile(T_2, N_2, S_2, E_1, E_2)$$
$$\rightarrow correct(1, \boldsymbol{SJ})$$

$$succ(\boldsymbol{SI}, \boldsymbol{I}) \wedge correct(\boldsymbol{I}, 1) \wedge f(\boldsymbol{I}, 1, T_1) \wedge f(\boldsymbol{SI}, 1, T_2)$$
$$\wedge tile(T_1, N_1, S_1, W_1, E_1) \wedge tile(T_2, S_1, S_2, W_2, E_2)$$
$$\rightarrow correct(1, \boldsymbol{SJ})$$

$$length(\boldsymbol{L}) \wedge correct(\boldsymbol{L}, \boldsymbol{L}) \rightarrow \mathsf{attack}$$

Table 3. Typed TASLan encoding of the Tiling problem.

choose for each positive predicate of the rule a matching predicate from $HC(S_i)$ and check that the negative predicates are not contained in S_i (recall only explicit facts can occur negatively in a rule); check the equalities and inequalities and compute the appropriate successor state. Finally, accept if S_k contains $\mathsf{attack}()$.

This machine accepts iff an attack state is reachable in l steps or less. The runtime of this non-deterministic machine is bounded by $|U| \cdot N \cdot l \in O(2^{poly(N)})$ because computing the Horn closure costs at most $|U|$ steps, matching the rule to some choice of predicates at most N steps, and we have at most l such computations. Thus the machine has non-deterministic exponential time.

Bounded TASLan insecurity is NEXPTIME-hard. We encode the NEXPTIME-complete Tiling problem [13]: A tile $T = (N, S, E, W)$ is a four-tuple of integers. As input we are given a collection $C = \{T_1, \ldots, T_n\}$ of tiles where $E_{T_i} = W_{T_{i+1}}$

for $1 \leq i < n$, as well as a size parameter $N \leq n$. The question is: exists a mapping $f : \{1, \ldots, N\}^2 \to C$ such that $E_{f(i,j)} = W_{f(i,j+1)}$ and $N_{f(i,j)} = S_{f(i+1,j)}$ for all $1 \leq i, j < N$ and $f(1, i) = T_i$ for $1 \leq i < n$?

We give the following encoding into TASLan (using the typed model). Let the search bound $l = N^2$. Let $k = \lceil \log_2 N \rceil$ be the number of bits to represent a counter up to N. We write in the following the vector notation $\boldsymbol{N}$ to represent an integer N as a binary vector $\langle N_{k-1}, \ldots N_0 \rangle$; we also use this notation for vectors of binary variables, and $\boldsymbol{bin}$ for the respective type.

We use the following symbols and types:

Symbol	Type	Intuitive Meaning
$tile$	$(tileID, int, int, int, int)$	Description of Tile
f	$(\boldsymbol{bin}, \boldsymbol{bin}, tileID)$	representing function value
$correct$	$(\boldsymbol{bin}, \boldsymbol{bin})$	f correct up to here
$succ$	$(\boldsymbol{bin}, \boldsymbol{bin})$	"$succ(N, N+1)$"

Note that the int type is just pro-forma, being defined by the set of constants used in the tiles itself. The constants and variables we use have the appropriate types. Let $t_1, \ldots, t_n$ be constants that identify the tiles $T_1, \ldots, T_n$.

The TASLan specification of initial state, transition rules, and Horn theory is found in Table 3. This specification non-deterministically chooses any tiling f and then checks using the Horn clauses whether this is indeed a correct solution. A state is an attack iff we have found a solution, thus an attack state is reachable iff a solution exists.

As this reduction is polynomial in the size N of the original problem, we have shown that bounded TASLan insecurity is at least as hard as all NEXPTIME problems. Together with the containment in NEXPTIME we thus have NEXPTIME completeness. □

5 Towards a Full ASLan

We have so far neglected some features of the original ASLan that seem less central to us. Here we briefly discuss how to (partially) support these features as well.

"Wildcard" Horn Clauses For Horn clauses $\forall \boldsymbol{X} : S \to P$ we had previously required $fv(P) \subseteq fv(S)$ (and $fv(S) \subseteq \boldsymbol{X}$) so the right-hand side cannot introduce new variables. Thus prevents clauses like $\forall X :\to p(X, X)$. Dropping this restriction causes a slight problem for the symbolic approach, because this may lead to non-termination of Horn closure (since this introduces new variables). Further this can destroy the well-formedness condition for the symbolic intruder deduction (because the new variables are not depending on a choice of the intruder). This limitation can be overcome with a special predicate $isBeta(\cdot)$ for new variables of type β. An example is found in the proof of NEXPTIME-hardness in Theorem 3.

A way to deal with this, is shown in the proof of NEXPTIME-hardness in Theorem 3: whenever the conclusion P of a Horn clause contains a variable X of a basic type β that is not introduced in the premises, then add another premise $isBeta(X)$. We then only need to ensure $isBeta(t)$ indeed holds for all terms t of type β. This is straightforward for basic types: we can explicitly write this in the initial state for all β-constants that occur in the specification, and every state transition that creates a fresh constant of type β shall have the type declaration on the right-hand side. For a variable X of a composed type $f(\boldsymbol{\tau})$, replace X with $f(\boldsymbol{X})$ for fresh variables $\boldsymbol{X}$ of the respective subtypes. Thanks to the typing result, these transformations of the specification are without loss of attacks.

Subtypes The original ASLan allows the declaration of subtypes, e.g. honest as a subtype of agent. Such an example could be modeled in TASLan by having only the basic type agent, and a special predicate honest (of type agent) that holds true for all those agents that are honest. A similar encoding is possible for two composed types τ_1 and τ_2 that differ only in a basic subtype.

Mappings For many problems it is helpful to have some function symbols to express mappings such as $sk(A, B)$ to denote the shared key between agents A and B, but of course the resulting key is then of type $sk(agent, agent)$ and thus with terms of a basic type *symkey* to represent symmetric keys. More generally, the problem is to model a function f of type $\boldsymbol{tau} \to \tau_0$ where $\tau_0 \neq f(\boldsymbol{tau})$. A way to achieve this is the use of a new predicate to represent the function, e.g. in the example $sk' : (agent, agent, symkey)$. We then need to take care of an appropriate constant for the function result, e.g. the symmetric key here. Also this encoding has its limitations: for instance a function like $s : \beta \to \beta$ cannot be injective (as this would lead to an infinite type).

Algebraic Properties One may consider algebraic properties to support some cryptographic primitives. This quickly rules out many methods, e.g. when the equivalence class of a term gets infinite. We just hint that for some algebraic properties the symbolic methods work [10].

FOLTL Goals We have considered only state-based safety properties, while ASLan allows for the specification of FOLTL goals, i.e. first-order logic extended with the temporal operators of linear temporal logic. Again this logic gives rise to undecidable problems in general. Borrowing from the arguments of Theorem 3, we can however identify a decidable fragment that fits with the TASLan approach. The idea is that typed TASLan for a bounded-length trace gives a finite universe of predicates (because also the number of fresh constants that can be created is bounded). When checking safety properties, considering finite traces is no restriction. When checking non-safety properties (e.g. for resilient channels [3]) common methods also need to consider finite state spaces (so that all infinite traces go in loops through the state space), in which case it is also

not a restriction to limit the number of fresh constants. When we have a finite universe, then we can reduce problem into a propositional LTL problem.

We have left this out of our main presentation since many verification methods based on abstract interpretation and symbolic constraints do not combine well with FOLTL goals, for instance a goal like $G(\mathsf{ik}(s) \rightarrow \mathsf{ik}(t))$ implies negative intruder knowledge constraints that cannot be directly handled.

6 Conclusions

ASLan is a specification language that integrates Horn clauses with transition systems, and this combination in the specification language gives a particular expressive power: we can formulate a transition system with immediate evaluations for every state. The typical application is the interaction between the work-flow of a distributed system and its access control policies, see the AVANTSSAR case studies for a large class of security-relevant systems [2]. A completely different application to combine immediate evaluations with transitions is in our recent work to analyze security of virtualized infrastructures [8]. Here we model a network representing a virtualized infrastructure that can change due to actions of honest agents and intruder. The Horn clauses can be used to make evaluations on the network in each state, e.g. between which nodes information flow is possible.

In this paper we have reviewed the syntax and semantics of ASLan, giving a conceptually simpler account than previous definition [4]. We have extended the concepts of symbolic transition systems to ASLan as a logically sound basis for constraint-based model-checking.

We have defined the fragment TASLan by the requirement that messages and predicates of different intended types must have sufficiently different formats so they cannot be confused. To a large extend, such disambiguations are good engineering practice anyway, and we can exploit this to obtain a class of specifications that is better to tackle with automated methods, while maintaining the powerful concept of combining transition systems with immediate evaluations.

We have built a decision procedure for bounded-length TASLan (or a semi-decision procedure for unbounded length) extending the constraint-based "lazy intruder" approach [15, 16, 6] to support the combination with Horn deduction constraints.

We show that, when an attack exists, the lazy intruder will find a well-typed attack, and thus we have a generalization of several typing-results [11, 7, 1]: for TASLan specification, we can safely restrict the verification to a typed model. This enables methods that cannot deal with an infinite universe of intruder-generated messages or only under great difficulty. Seen another way, the typed model simplifies the undecidable logic-programming problem induced by the Horn clauses to a decidable Datalog problem.

On the conceptual side, we show that the problem whether a TASLan specification has an attack for a bounded number of transitions is NEXPTIME complete.

Despite the high complexity class, first experiments with extending the tool OFMC [6] demonstrate that the method is feasible for many practically relevant problems: a first prototype successfully analyzes 70 of the 142 ASLan specifications of the AVANTSSAR library [2] in under 8 minutes.

References

1. M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *FSTTCS*, pages 376–387, 2007.
2. A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar, G. Erzse, S. Frau, M. Minea, S. Mödersheim, D. von Oheimb, G. Pellegrino, S. E. Ponta, M. Rocchetto, M. Rusinowitch, M. Torabi Dashti, M. Turuani, and L. Viganò. The AVANTSSAR Platform for the Automated Validation of Trust and Security of Service-Oriented Architectures. In *Proceedings of TACAS*, LNCS 7214, pages 267–282, 2012.
3. A. Armando, R. Carbone, and L. Compagna. LTL Model Checking for Security Protocols. In *Proceedings of CSF20*. IEEE Computer Society Press, 2007.
4. The AVANTSSAR Project: Deliverable 2.3: ASLan (final version), 2010. Available at `www.avantssar.eu`.
5. F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *PODS*, pages 1–15, 1986.
6. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
7. B. Blanchet and A. Podelski. Verification of cryptographic protocols: tagging enforces termination. *Theor. Comput. Sci.*, 333(1-2):67–90, 2005.
8. S. Bleikertz, T. Groß, and S. Mödersheim. Automated verification of virtualized infrastructures. In *CCSW*, pages 47–58, 2011.
9. I. Cervesato, N. A. Durgin, J. C. Mitchell, P. Lincoln, and A. Scedrov. Relating strands and multiset rewriting for security protocol analysis. In *CSFW*, pages 35–51, 2000.
10. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *FST TCS'03*, LNCS 2914, pages 124–135, 2003.
11. J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of CSFW'00*. IEEE Computer Society Press, 2000.
12. A. V. Hess and S. A. Mödersheim. A Typing Result for Stateful Protocols. In *CSF*, 2018. To appear, pre-print available at `http://compute.dtu.dk/~samo`.
13. D. S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 67–161. Elsevier, 1990.
14. C. Meadows. Analyzing the needham-schroeder public-key protocol: A comparison of two approaches. In *ESORICS*, pages 351–364, 1996.
15. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS'01*, pages 166–175. ACM Press, 2001.
16. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theor. Comput. Sci.*, 1-3(299):451–475, 2003.