A Typing System for AIF Set-Based Abstraction

Andrew Habib

Kongens Lyngby 2015

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

Formal methods are of wide use nowadays due to the robustness and strength of results obtained through them. Communication and cryptographic protocols are widely and critically used in various forms when interacting and dealing with software and communication systems. Therefore, verification of security protocols is one of the main areas for applying formal techniques to discover errors and/or prove protocols to be correct. Different models have been suggested and applied to this problem.

Traditional approaches to formal verification of security protocols reach a limit when dealing with complex systems that allow revocation of keys or access rights for example. This led to a new technique called AIF Set-Based Abstraction [Möd10] that is capable of verifying such complex systems. However, this new approach is undecidable and verification of a protocol is not guaranteed to terminate.

The purpose of this masters thesis is to extend the AIF approach wit a type system to obtain a decidable verification problem. We define our type system and prove soundness and decidability results of our typed model. Moreover, we extend the existing AIF translator tool so that it can translate protocol specifications under the new typed AIF. Finally, we add an extra feature to the AIF tool by integrating a fixedpoint module from the Open-Source FixedPoint Model-Checker (OFMC) [BMV05].

Preface

This thesis is submitted in partial fulfillment of the requirements for a M.Sc. in Engineering, Security and Mobile Computing (NordSecMob Masters Program). It was prepared at Denmark Technical University Compute Department (DTU Compute) and it contains work done from February to July 2015. The thesis has been made solely by the author; most of the text, however, is based on the research of others and discussions with the supervisor, and we have done our best to provide references to sources where applicable.

As part of the mobility requirement of the NordSecMob program, I spent my first semester (Fall 2013) at the Norwegian University of Science and Technology (NTNU). I studied the rest of my degree at Denmark Technical University (DTU).

This thesis was supervised by professor Sebastian Mödershiem (DTU) and co-supervised by professor Peter Herrmann (NTNU).

Lyngby, 16-July-2015

Andrew Habib

Acknowledgements

I would like to express my appreciation and gratitude to my supervisor, professor Sebastian M. His sound and thoughtful advice along with interesting and mind-challenging discussions made my masters thesis experience very enjoyable and helped my develop a mature perspective in research. Working with him was pleasure and I would do it again.

A special thank you goes to the love my life, Lydia, for her continuous support, encouragement and endurance of our long-distance relationship. Also, I express my sincere gratitude to my lovely family and dear friends for the unstoppable support, unconditional love and their persistent prayers.

Above all, I give all thanks and praise to God for guiding and leading every step of my life, including my masters degree time and the life yet to come.

Contents

Chapter 1

Introduction

Security protocols are set of steps that describe interaction between two or more parties to ensure the protection of valuable data in a hostile environment. Protection of data is multifaceted and spans various aspects such as confidentiality, authenticity, integrity and others. Because security protocols are ubiquitous: used everywhere in wireless networks, e-commerce, communication, and others; ensuring they are well-designed and correctly implemented is very important. Starting early nineties, verification of security protocols has become of increasing importance as history showed that informal arguments of the security of a protocol specification is strongly insufficient. For example, the famous long thought to be secure Needham-Schroeder Public Key (NSPK) protocol [NS78] which was proposed in 1978 was discovered to be vulnerable to a man-in-the-middle attack in 1995 by G. Lowe [Low95].

In fact, verifying security protocols is not easy and indeed notoriously difficult for various reasons. Usually, security protocols are designed under the strong assumption that communication takes place in a completely hostile environment where an adversary has full control over the underlying communication medium. He can intercept, block, forge, alter or replay messages on their way to the lawful participants. Moreover, the cryptographic primitives used in security protocols are subtle and are difficult to implement correctly. Also the fact that security protocols are executed simultaneously with multiple number of sessions that involve many players makes it very hard to prove the correctness of the protocols.

Besides, errors in security protocols cannot be detected by normal "functional" testing; but rather they are discovered in the presence of an attacker during actual protocol execution. Finally, the increasing complexity and heterogeneity of communication systems and web services makes the verification of security protocols more cumbersome and error-prone.

Hence was the need for more formal arguments and techniques to prove the correctness of security protocols. Over the past 20 years, researchers have focused on devising techniques and tools for formal verification of security protocols. Theses efforts by the various research communities yielded many breakthroughs and tangible achievements. [BCM11] provides a survey of some of the state-of-the-art tools for formally verifying security protocols.

1.1 Motivation

Verification of security protocols is an active area of research due to the reasons mentioned above. Many tools and techniques have proven to be efficient and correct in different real-life scenarios. However, as we discuss in the following background chapter, there is much room for improvement at various levels; specially regarding the general problem of undecidability of the security of a protocol specification.

In this work, we explore one of the novel techniques for protocol verification: AIF - Abstraction by Set-Membership [Möd10]. We extend the original AIF model with a type system. This allows us to reach new decidability and soundness results. We also implement our proposed type system as an extension to original AIF translator as we explain later. Moreover, we implement a fixed-point computation module similar to the one in the Open-source Fixed-point Model Checker for security protocols (OFMC) tool [BMV05] as an add-on extension so that the AIF tool could be used as a stand-alone verifier; beside being a translator of abstract model to the syntax of other verifiers.

Chapter 2

Background

In this chapter, we briefly introduce the important and relevant parts in the literature of protocol verification. Not only does this motivate our work, but also it puts our contribution in perspective from other relevant work.

2.1 Protocol Verification

The literature of protocol verification proposes two main models for verifying security protocols:

1. *The Computational Model*

   This model is mainly considered by cryptographers and was first introduced by Goldwasser, Micali and others early eighties [GM84]. In the computational model, messages in protocol interactions are bit-strings, cryptographic primitives are functions from bit-strings to bit-strings and the attacker is any probabilistic polynomial-time Turing machine. This model is considered to be closer to actual execution of protocols but automating verification of protocols under this model is quite hard. [Bla08] discusses some ideas related to proof automation in the computational model.

2. *The Symbolic Model*

   This model is due to Needham and Schroeder in 1978 as well as Dolev and Yao in 1981 and usually called the Dolev-Yao model [NS78, DY83]. It relies on the assumption of perfect cryptography where cryptographic primitives are considered black boxes. Messages are terms over an algebra of symbols that model the cryptographic primitives. The attacker can indefinitely compute messages using these primitives and the model permits adding equations between the primitives. Verifying security protocols under the symbolic model is more desirable since the abstraction provided by this model simplifies the verification process greatly.

Because the symbolic model is simpler compared to the computational model, it has been the main focus of research work. In fact, the symbolic model yielded many successful results in automatic protocol verification. This why we focus our work here on the symbolic model.

The idea of verifying security protocols in the symbolic model is to compute the set of messages (terms) an attacker may ever know. If a message does not belong to the set of messages known by the attacker, it is considered to be secret. It is not as easy as it sounds because the adversary can compute messages unboundedly. In addition, the number of protocol executions (sessions) is infinite. Therefore, the decision problem of whether a message is in the intruder knowledge or not is in general undecidable. That said, different techniques have been suggested in the literature to tackle this undecidable problem.

2.1.1 Standard Model Checking

Known as finite-state analysis, model checking in simple terms is: given a model of a system, automatically and exhaustively check whether this model has some desired properties (if the model meets some specifications) [SS98]. Since it is a "finite-state" analysis, conventional model checking requires bound on both the messages size the intruder can compute and the number of sessions. Tools like FDR and the SATMC backend of AVISPA utilize this approach.

With a bound only on the number of sessions, model checking still works and the security problem is decidable under reasonable assumptions: [RT01] shows that protocol *insecurity* in NP-complete. The main idea used by various tools is to avoid naive enumeration of the infinite state space and rather use a symbolic constraint-based approach. A novel and successful tool that uses the *lazy intruder* model is OFMC [BMV05], which is available as a stand-alone tool and also as a back-end for AVISPA [ABB+05].

In general, standard model checking under bounded number of sessions reveals several attacks on security protocols. However, the absence of attacks does not necessarily mean the protocol is safe.

2.1.2 Over-Approximation

This technique is used when the verification of a security protocol is desired for unbounded number of sessions. Over-approximation is very handy when verification (proving the security of a protocol) is preferred over falsification (finding an attack). However, the security decision problem remains undecidable in general. Since developing automatic tools that are guaranteed to terminate for an undecidable problem is not possible, different approaches are taken.

- Some tools such as Maude-NPA [EMM09] just allows non termination.
- Other tools relies on user-specified lemmas to guide the security proof, for example using the interactive theorem prover Isabelle [Pau94].
- TA4SP [Boi] uses tree-automata and ProVerif [BCAS10] uses Horn clauses to over-approximate the intruder knowledge but there is no guarantee on termination.
- A different approach proposed by the OFMC Fixedpoint module where data and control abstractions are used to over-approximate the intruder knowledge [BMV05]. Moreover, automatic refinement of the abstraction is done in case of reporting attacks. This method is restricted to protocol specifications that do not use negative set conditions.

Though not decidable in general, verification for unbounded sessions using TA4SP, ProVerif or OFMC Fixedpoint could be guaranteed to terminate using typing or tagging [BP03].

Over-approximations in general do not consider a state transition system, but rather just the set of state independent derivable facts; for instance: the intruder knowledge. The main advantages of over-approximation models are: verification for unbounded number of sessions and the proof of security of the over-approximation model guarantees the security of the concrete model. Moreover, since over-approximation abstracts protocols to just set of derivable facts, the verification problem can be encoded into Horn clauses. And hence, general first-order Horn clauses solvers such as SPASS [WDF$^+$09] or Z3 [DMB08] could be used to solve the verification problem.

2.2 Over-Approximation by Set-Membership

Over-approximations has two advantages: the first one is that the abstractions yield false positives. This problem is in general solved by either user-assisted or automatic refinement like the OFMC Fixedpoint [BMV05] discussed above. The second disadvantage is that such abstraction techniques are not suitable for the verification of complex protocols that involve databases of keys or access rights where revocation is possible.

Indeed such systems challenge the over-approximation techniques as the set of derivable true facts does not grow monotonically with the execution of protocols. For example, in a client-server scenario, the server may revoke a previously valid key at some agent. Also, access rights for normal systems are granted and revoked continuously. The notion of revocation is problematic because the fact of revoking a key for instance, implies that the set of true facts of valid keys is smaller. This non-monotonic behaviour cannot be modeled in standard Horn clauses abstraction approaches as deduction is monotonic. In other words, adding new facts (like key revocation) should never result in fewer true facts.

To solve this problem, a novel technique proposed by Mödersheim in [Möd10] tackles this issue by a different kind of abstraction of fresh data. The method: AIF Set-Based Abstraction considers a protocol model where participants maintain (databases) of fresh data and their corresponding contexts: owners and status for example. Abstraction of freshly created data is according to their status and membership in the databases of the different participants.

The method follows the usual idea of abstracting the infinite number of fresh constants created in the concrete model to finitely many equivalence classes in the abstract model (under the condition that there is finitely many participants). In complex systems with databases, an abstract value - say a key K - may change its set membership during the protocol execution for instance by being revoked at some agent. This behaviour is captured by an unusual rule, called *term-implication rule* of the form: $\phi \rightarrow k \twoheadrightarrow k'$. This rule indicates that if clauses in ϕ holds, then for every context $g[\cdot]$: $g[k]$ implies $g[k']$. This type of rule is different from rewrite rules because $g[k]$ is not replaced by $g[k']$ but both $g[k]$ and $g[k']$ hold. Also it is not similar to algebraic equations like $k \approx k'$ because $g[k']$ doesn't necessarily imply $g[k]$. The technique shows how this new term-implication rules could be encoded into standard Horn clauses.

We start by looking into the important details of the AIF Set-Based Abstraction. This will serve as a stepping-stone in extending the original AIF model with a type system.

Chapter 3

AIF

In this chapter, we present summary of the original AIF model and its formal definition from [Möd10]. We do not claim any contribution in this part; but rather it serves as the foundation for our modified AIF.

AIF specification of a protocol is a set of state transition rules that describe the protocol execution steps. A rule consists of a *left hand side (LHS)* and a *right hand side (RHS)*. The LHS specifies the preconditions that a state must meet so that the rule could be applied to a state. Absence of the RHS of a rule indicates that this rule can be taken at any state because it has no preconditions. The RHS of a rule describes the effect (change) that result from applying the rule to a state. The high level form of the AIF transition rules is:

$$LHS \Rightarrow RHS$$

where $\Rightarrow$ could take a variant form of $=[F]\Rightarrow$ to indicate that the transition rule creates a new fresh value F. The following formal definition of AIF gives more details.

3.1 Formal Definition of AIF

Definition 3.1 Messages in AIF are terms over a signature $\Sigma \cup \mathfrak{A}$ and a set of variables $\mathcal{V}$ where Σ is finite, $\mathcal{V}$ is countable and $\mathfrak{A}$ is a countable set of constant symbols. Σ, $\mathfrak{A}$ and $\mathcal{V}$ are non-empty pairwise disjoint. Moreover, $\mathcal{V}_{\mathfrak{A}} \subset \mathcal{V}$ is a set of variables that can only be substituted by constants from $\mathfrak{A}$. The set $\mathcal{T}_{\mathfrak{A}} = \mathfrak{A} \cup \mathcal{V}_{\mathfrak{A}}$ is called the set of all *abstractable symbols*.

Definition 3.2 Σ_f is a finite signature of fact (predicate) symbols that is disjoint from $\Sigma, \mathfrak{A}$ and $\mathcal{V}$. A *fact* is a term that has the form $g(t_1, \ldots, t_n)$ where $g \in \Sigma_f$ is a fact symbol of arity n and t_i are messages.

Definition 3.3 A positive (negative) *set condition* is of the form $t \in M$ ($t \notin M$) where $t \in \mathcal{T}_{\mathfrak{A}}$ and M is a *set expression*: a ground term with no occurrence of symbols from $\mathcal{T}_{\mathfrak{A}}$.

Definition 3.4 A *state* S is a finite set of facts and positive set conditions.

Definition 3.5 A *transition rule* r has the form

$$LF \cdot S_+ \cdot S_- \; =[F]\Rightarrow \; RF \cdot RS$$

where LF and RF are sets of facts, S_+ and RS are sets of positive set conditions, S_- is a set of negative set conditions; and $F \subseteq \mathcal{V}_{\mathfrak{A}}$. In addition, a requirement is enforced by definition that $vars(RF \cdot RS \cdot S_-) \subseteq F \cup vars(LF \cdot S_+)$ and $vars(S_-) \cap F = \emptyset$. Another requirement is that for each $t \in \mathcal{T}_{\mathfrak{A}}$ that occurs in S_+ or S_- it must also occurs in LF; and for for each $t \in \mathcal{T}_{\mathfrak{A}}$ that occurs in RS it must also occurs in RF.

The left-hand side of a transition rule describes states to which this rule could be applied, and the right-hand side of the rule denotes the changes to the state after applying the transition rule.

Definition 3.6 A transition rule r can be applied to state S to result in state S' written as $S \Rightarrow_r S'$ if and only if there exists a grounding substitution σ (for all variables in r) such that

- $(LF \cdot S_+)\sigma \subseteq S$
- $S_-\sigma \cap S = \emptyset$, $S' = (S \setminus S_+\sigma) \cup RF\sigma \cup RS\sigma$
- $F\sigma$ are fresh constants from $\mathfrak{A}$ (they do not occur in S or any rule r that we consider)

Definition 3.7 A state S is *reachable* using the set of transition rules R if and only if $\emptyset \Rightarrow_R^* S$ where $\emptyset$ is the initial state, $\Rightarrow_R$ is the union of $\Rightarrow_r$ for all $r \in R$; and $.^*$ is the reflexive transitive closure.

Now, we can introduce our modification on the original AIF specification in the following chapter. For comprehensive details about AIF, we refer the reader to [Möd10].

3.2 Definition of the Abstraction

The idea of abstraction by set membership is that fresh data is abstracted according to its set membership in the used sets. Suppose we have a protocol specification that uses three sets s_1, s_2 and s_3. All elements that are contained in s_1 but not in s_2 and s_3 could be abstracted by $val(1, 0, 0)$ which represents the entire equivalence class of those elements.

To capture the non-monotonic behaviour of the set of true facts, the AIF model introduce the following notion. When the set membership of an element changes from the abstract value a to a', then for every derivable fact $g[a]$ also $g[a']$ is derivable. This is expressed using the rule: $g[a] \Longrightarrow g[a']$ for every context $g[\cdot]$.

3.2.1 The Abstraction

Take a set of AIF rules that uses the ground terms $s_1, \ldots, s_N$ in set conditions $t \in s_i$ ($t \notin s_i$). For a state S, define the function $abs_S : \mathfrak{A} \to val(\mathbb{B}^N)$ such that $abs_S(c) = val(b_1, \ldots, b_n)$ where b_i is true iff $(c \in s_i) \in S$. This definition gives an equivalence relation (parametrized by S) on $\mathfrak{A} : c \equiv_S c'$ iff $abs_S(c) = abs_S(c')$.

3.2.2 Term Implication Rules

This is the the kind of rule which capture the abstraction of constants that change their set-membership.

Definition 3.8 A *term implication rule* has the form

$$\frac{P_1 \ \ldots \ P_n}{s \twoheadrightarrow t}$$

for facts (predicates) P_i and $vars(s) \cup vars(t) \subseteq \bigcup_{i=1}^{n} vars(P_i)$.

An *implication rule* is either a term implication or a Horn clause.

For implication rules, define a function that, given a set Γ of facts, yields all derivable facts from Γ by one rule application:

$$\left[\!\left[\frac{\phi_1 \ \dots \ \phi_n}{\phi} \right]\!\right](\Gamma) = \{\phi\sigma \mid \phi_1\sigma \in \Gamma \wedge \dots \wedge \phi_n\sigma \in \Gamma\}$$
$$\left[\!\left[\frac{\phi_1 \ \dots \ \phi_n}{s \twoheadrightarrow t} \right]\!\right](\Gamma) = \{C[t\sigma] \mid C[s\sigma] \in \Gamma \wedge \phi_1\sigma \in \Gamma \wedge \dots \wedge \phi_n\sigma \in \Gamma\}$$

where $C[\cdot]$ is a *context* i.e. a "term with a hole". So $C[t]$ means filling this hole with the term t. The *least fixed-point* of a set of implication rules R, denoted $LFP(R)$ is defined as the smallest (least) set Γ that is closed under $[\![r]\!]$ for each $r \in R$.

3.2.3 Translation to Abstract Rules

This section briefs on how the model translates standard 'concrete' transition rules (written in the AIF language and uses normal 'real' sets) to implication rules of the abstract model (which operates on the abstract equivalence classes of sets).

Definition 3.9 Let the concrete transition rule be

$$r = LF \cdot S_+ \cdot S_- \ \text{=}[F]\Rightarrow\ RF \cdot RS$$

Let $\mathcal{T}_{\mathfrak{A}}(r)$ be the symbols from $\mathcal{T}_{\mathfrak{A}}$ which occur in r. Define for each $t \in \mathcal{T}_{\mathfrak{A}}$ and for each $1 \leq i \leq N$:

$$L_i(t) = \begin{cases} 1 & \text{if } t \in s_i \text{ occurs in } S_+ \\ 0 & \text{if } t \notin s_i \text{ occurs in } S_- \\ X_{t,i} & \text{otherwise} \end{cases}$$
$$R_i(t) = \begin{cases} 1 & \text{if } t \in s_i \text{ occurs in } RS \\ X_{t,i} & \text{if } L_i(t) = X_{t,i} \text{ and } t \notin F \\ 0 & \text{otherwise} \end{cases}$$

Let $X_{t,i} :: \mathbb{B}$ be variables that do not occur in r. Define:

$$L(t) = val(L_1(t), \dots, L_N(t))$$
$$R(t) = val(R_1(t), \dots, R_N(t))$$

The *abstraction of rule r* is:

$$\bar{r} = LF\lambda \to RF\rho \cdot C$$

for substitutions λ and ρ and term implications C:

- $\lambda = [t \to L(t) \mid t \in \mathcal{T}_{\mathfrak{A}}(r)]$
- $\rho = [t \to R(t) \mid t \in \mathcal{T}_{\mathfrak{A}}(r)]$
- $C = \{t\lambda \twoheadrightarrow t\rho \mid t \in \mathcal{T}_{\mathfrak{A}}(r) \setminus F\}$

Further, the original paper AIF paper shows how the term implication rules $s \twoheadrightarrow t$ can be encoded into Horn clauses. But we opt not to include this part here. You can check [Möd10] for more details.

3.3 The AIF Language

A protocol specification is written in this language before the abstraction could be created. The AIF language is a variant of the AVISPA Intermediate Format (AVISPA IF) [ABB+05] with the following modifications.

AIF allows variables of type *untyped*. Such variables are used to denote arbitrary massages. Syntactically, the AIF specification language permits these *untyped* variables to appear anywhere whether on the LHS or the RHS of any protocol transition rule. But, they cannot be used in creating fresh constants. In fact, the original specification requires and enforces that fresh variables $\mathcal{V}_{\mathfrak{A}}$ can be substituted only by constants from the countable set $\mathfrak{A}$ of fresh constants.

In addition, the AIF language supports *enumeration variables* which are variables that range over given sets of constants. For example, one can define

$$A, B : \{a, b, s, i\}$$

$$Status : \{valid, revoked\}$$

for the set of agents and status of used keys. This declarations means that the variables A, B and $Status$ could only be instantiated by the set of constants: $\{a, b, s, i\}$ and $\{valid, revoked\}$ respectively. Such variables come handy in many ways. They could be used to specify sets:

$$ring(A, Status)$$

denotes the database (set) of keys that every agent in A maintains for every corresponding status. This example means that we have $4 * 2 = 8$ sets because every agent maintains two databases: one for valid keys and one for revoked keys. Also, enumeration variables could be used with universal quantification in set conditions:

$$\forall A, Status.K \notin ring(B, Status)$$

which means that K does not occur in any of the sets obtained by expanding all values of the enumeration variables. Therefore, this set condition expands to 8 negative set conditions. Finally, AIF rules could be parametrized over enumeration variables. For instance, one can write

$$\lambda A. \Rightarrow iknows(A)$$

to specify that the intruder knows all agents names. The meaning of $\lambda X.r$ where X is an enumeration variable and a rule r is that $\{r[X \mapsto v] \mid v \in V\}$ for V enumeration values declared for X.

Now, that we have established the necessary background and foundation for our modification, in the following chapter we present our proposed typed AIF.

Chapter 4

Typed AIF

The original AIF model includes an implicit notion of type that is intended for the set of *abstractable symbols* $\mathcal{T}_{\mathfrak{A}} = \mathfrak{A} \cup \mathcal{V}_{\mathfrak{A}}$ of constants and variables. The specification stipulates that variables in $\mathcal{V}_{\mathfrak{A}}$ can only be substituted by constants from $\mathfrak{A}$ which are abstracted later.

However, the intruder capabilities given in an AIF specification, which is based on the standard Dolev-Yao attacker model, specify unbounded intruder deduction rules. For example, an attacker may infinitely concatenate messages from his knowledge to produce new ones. Now, to bound the intruder knowledge - which is unbounded due to untyped specification in AIF - so as to achieve a decidable verification problem of the security of an AIF specification, we need to modify the AIF specification to be able to enforce strong types on messages and terms that may ever occur in a protocol interaction.

Therefore, we begin this chapter by formally defining a type system that we use to enforce strong types on AIF specification. After that, we discuss how to use this type system to bound the intruder deduction capabilities which will lead to a decidable verification problem for the security of an AIF protocol specification.

4.1 Definition of a Typing System

To be able to define a strong type system on all messages and their parts (terms), we first need to define the subterm relation $\sqsubseteq$.

Definition 4.1 Let $\mathcal{T}_{\Sigma\cup\mathfrak{A}}(\mathcal{V})$ be the set of all possible terms. $(\sqsubseteq, \mathcal{T}_{\Sigma\cup\mathfrak{A}}(\mathcal{V}))$ is a partial order *subterm relation* on the set of terms:

For all $s, t, u \in \mathcal{T}_{\Sigma\cup\mathfrak{A}}(\mathcal{V}))$

- Reflexive: $t \sqsubseteq t$
- Transitive: $(s \sqsubseteq t) \wedge (t \sqsubseteq u) \Rightarrow s \sqsubseteq u$
- Anti-symmetric: $(s \sqsubseteq t) \wedge (t \sqsubseteq s) \Rightarrow s = t$

Moreover, for all function symbols $f \in \Sigma$ of arity n, we have $t_{i\in\{1,\ldots,n\}} \sqsubseteq f(t_1, \ldots, t_n)$.

To enforce strong typing on AIF specification, we need to distinguish atomic types from composed ones. Atomic types are either *atom* or *untyped* while *composed* types are obtained by applying function symbols from Σ to terms of atomic or composed types. Our proposed type system assigns types to all terms in $\mathcal{T}_{\Sigma\cup\mathfrak{A}}(\mathcal{V}))$ according to the following type function.

Definition 4.2 Let $\Psi : \mathcal{T}_{\Sigma\cup\mathfrak{A}}(\mathcal{V}) \to \mathcal{T}_{\Sigma\cup\{atom,untyped\}}$ be a total function that maps all terms in $\mathcal{T}_{\Sigma\cup\mathfrak{A}}(\mathcal{V})$ to types $\mathcal{T}_{\Sigma\cup\{atom,untyped\}}$. For all abstractable symbols $\mathcal{T}_{\mathfrak{A}} = \mathfrak{A} \cup \mathcal{V}_{\mathfrak{A}}$ let $\Psi : \mathcal{T}_{\mathfrak{A}} \to \{atom\}$ and for all variables $\mathcal{V}$ let $\Psi : \mathcal{V} \to \{atom, untyped\}$. Also, for a term $t = f(t_1, \ldots, t_n)$ where $f \in \Sigma^n$, define $\Psi(t) = f(\Psi(t_1), \ldots, \Psi(t_n))$. We write $t : \tau$ to denote that a term t has type τ.

This means that we require all constants and variables in $\mathcal{T}_{\mathfrak{A}}$ to be of type *atom* while variables in $\mathcal{V}$ could either be of type *atom* or *untyped*. And for any composed term, its type is determined through the types of its subterms. For example, if $x, y : atom$ and $pair(\cdot,\cdot) \in \Sigma^2$ then $\Psi(pair(x, y)) = pair(atom, atom)$.

We also define a subtype relation $\preccurlyeq$ on the set of possible types in the following manner.

Definition 4.3 Let $\mathcal{T}_{\Sigma\cup\{atom,untyped\}}$ be the set of all possible types of terms. $(\preccurlyeq, \mathcal{T}_{\Sigma\cup\{atom,untyped\}})$ is a partial order *subtype relation* on the set of types where $\preccurlyeq$ is the least relation such that

- $\tau \preccurlyeq \mathit{untyped}$ for all $\tau \in \mathcal{T}_{\Sigma \cup \{atom, untyped\}}$
- For $\tau_1, \ldots, \tau_n, \tau'_1, \ldots, \tau'_n \in \mathcal{T}_{\Sigma \cup \{atom, untyped\}}$ we write $\tau_1, \ldots, \tau_n \preccurlyeq \tau'_1, \ldots, \tau'_n$ iff $\tau_i \preccurlyeq \tau'_i$ for all $i \in \{1, \ldots, n\}$
- For every symbol $f \in \Sigma$ of arity n define $f(\tau_1, \ldots, \tau_n) \preccurlyeq f(\tau'_1, \ldots, \tau'_n)$ if $\tau_1, \ldots, \tau_n \preccurlyeq \tau'_1, \ldots, \tau'_n$.

Now, we have established the type system that serves as a ground to introduce the formal definition of the typed AIF specification.

4.2 Definition of Typed AIF

A term $t \in \mathcal{T}_{\Sigma \cup \mathfrak{A}}(\mathcal{V})$ is *typed* iff $\Psi(t) \in \mathcal{T}_{\Sigma \cup \{atom\}}$. In other words, a term is typed if it has no subterms (variables) of *untyped* type. Similarly, a fact $g \in \Sigma_f^n$ of the form $g(t_1, \ldots, t_n)$ is a *typed fact* iff $t_1, \ldots, t_n$ are all typed terms.

Definition 4.4 A substitution $\sigma_{_T}$ is *well-typed* iff $\forall x \in dom(\sigma_{_T}) : \Psi(x) = \Psi(\sigma_{_T}(x))$ or $\Psi(x) = \mathit{untyped}$.

Lemma 4.5 *If $\sigma_{_T}$ is a well-typed substitution, then $\Psi(t) \succcurlyeq \Psi(t\sigma_{_T})$ for all $t \in \mathcal{T}_{\Sigma \cup \mathfrak{A}}(\mathcal{V})$.*

Proof. By induction on terms.

Additionally, we say that an AIF specification is *message-bounded* if all variables can be instantiated only with variables of maximum depth.

We now come to the core of the typed AIF specification, namely *well-typed states* and *well-typed transition rules*.

Definition 4.6 A *state* $S_{_T}$ (cf. definition 3.4) is *well-typed state* if all of its facts and terms are typed.

Definition 4.7 A *transition rule* $S \Rightarrow_{r_{_T}} S'$ is *well-typed* if there exists a well-typed substitution that satisfies definition 3.6.

Using the set of well-typed transition rules $\Rightarrow_{R_{_T}}$, a well-typed state $S_{_T}$ is *reachable* if and only if $\emptyset \Rightarrow^*_{R_{_T}} S_{_T}$ where $\Rightarrow_{R_{_T}}$ is the union of $\Rightarrow_{r_{_T}}$ for all $r_{_T} \in R_{_T}$, $.^*$ is the reflexive transitive closure; and $\emptyset$ is the initial state.

Thus far in our discussion, we have not yet came to the point where we limit the intruder infinite composition capabilities. If an AIF specification makes all variables *untyped* (except for $\mathcal{V}_{\mathfrak{A}}$ which must be substituted only by constants from $\mathfrak{A}$ as the original AIF specification stipulates), then we have no restriction whatsoever and the typing system we introduced is just an explicit description of the original AIF. In fact, our model so far is exactly the same as the original untyped AIF; it is neither a restriction nor an extension in any sense. Therefore, our proposed typed AIF specification still incurs an undecidable verification problem [Möd10].

In the next section we explain how we utilize the aforementioned typed AIF specification to bound the intruder deduction rules.

4.3 Intruder Rules

The original untyped AIF uses an explicit type specifier *untyped* to denote variables used in the intruder deduction rules in the standard Dolev-Yao model. Such untyped variables are utilized to hold arbitrarily complex terms. For example, composition and decomposition of arbitrarily terms are given by:

- $iknows(M_1) \cdot iknows(M_2) \Rightarrow iknows(pair(M_1, M_2))$
- $iknows(pair(M_1, M_2)) \Rightarrow iknows(M_1) \cdot iknows(M_2)$

where M_1, M_2 are *untyped* variables, $pair(\cdot,\cdot) \in \Sigma^2$ is a two-arity function symbol and $iknows(\cdot) \in \Sigma_f^1$ a one-arity fact denoting intruder knowledge of a specific message (from now on, we use the fact symbol $iknows(\cdot)$ to refer to the intruder knowledge).

The problem with untyped composition rules is that they can be infinitely applied to messages in the current intruder knowledge to produce infinite and arbitrarily complex new messages. In other words, the closure of intruder knowledge under composition is infinite. An attacker who knows - for example - public constant symbols a and b and a fresh value N can indefinitely compose the terms:

$$\begin{aligned} &pair(a, b)\\ &pair(N, pair(a, b))\\ &pair(N, pair(N, pair(a, b)))\\ &\vdots \end{aligned}$$

This results in an infinite set of reachable states (and consequently infinite set of derivable facts) and hence the undecidability of the security of an AIF specification.

4.3.1 Intruder Composition

We begin this section by introducing the notion of *term depth*.

Definition 4.8 Let $t \in \mathcal{T}_{\Sigma \cup \mathfrak{A}}(\mathcal{V})$ be some term. Define *term depth* of t to be $depth(t) = 1$ if $t = x$ or $t = c$ for any variable x or constant c. For any composite term of the form $t = f(t_1, \ldots, t_n)$ where $f \in \Sigma$ is an n-ary function symbol define $depth(t) = Max\{depth(t_1), \ldots, depth(t_n)\} + 1$

To bound the intruder composition capabilities, we limit the attacker composition to typed terms of a maximum depth. This maximum depth is the largest depth of all possible terms in all protocol steps. To construct the well-typed attacker rules, we replace untyped composition rules with a finite set of well-typed rules with terms of - at most - the maximum depth.

Informally, we do so by traversing all normal protocol steps (i.e steps that do not specify the intruder deduction rules) and extract all possible typed terms that may ever occur in the intruder knowledge. For each such term, create a well-typed intruder composition rule that resembles the composition of this specific term.

To formulate the procedure of transforming untyped intruder composition rules to well-typed rules, we define the set:

Definition 4.9 Let r be a normal transition rule (not intruder deduction rule). Define:

$$P(r) = \{s \mid s \sqsubseteq t \ \wedge \ iknows(t) \text{ occurs in } r\}$$

the set of all possible terms (message patterns) that may ever occur in the intruder knowledge in a normal protocol transition rule r.

Moreover, we say that a message m is *ill-typed* if $m \notin P(R)$ where R is the set of normal transition rules.

The following procedure utilizes a distinction between two disjoint subsets of function symbols in Σ where the AIF specification allows declaring function symbols to be either *public*: Σ_{Pub} or *private*: Σ_{Prv} where Σ_{Pub} and $\Sigma_{Prv} \subset \Sigma$.

This distinction is very useful when modelling protocols where a private function is only accessible to some agent(s). For example, private key of Public-Private Key Scheme could be specified as $inv(PK)$ where $inv(\cdot) \in \Sigma_{Prv}$.

The set of well-typed intruder composition rules is constructed by the following simple algorithm:

Algorithm 1: Construction of bounded intruder composition rules

Data: Protocol Specification written in the AIF language and the set of declared public symbols Σ_{Pub}
Result: Set of well-typed intruder composition rules
begin
  for *each normal protocol transition rule* r **do**
    compute the set $P(r)$ of all possible message patterns;
    for *each* $p \in P(r)$ **do**
      if $p = f(t_1, \ldots, t_n)$ *for* $f \in \Sigma_{Pub}$ *and* $t_{i \in \{1,\ldots,n\}}$ *are well-typed* **then**
        add the rule:
        $iknows(t_1) \ldots iknows(t_n) \Rightarrow iknows(f(t_1, \cdot, t_n))$;

For example, for some protocol specification in AIF, let the set of public symbols be $\Sigma_{Pub} = \{pair(\cdot,\cdot), sign(\cdot,\cdot)\}$.

The transition rule:

$$r = iknows(PK) \cdot PK \in ring(H) \; \text{=}[NPK]\text{⇒} \; NPK \in ring(H) \cdot iknows(sign(inv(PK), pair(H, NPK)))$$

incurs the following set of message patterns:

$$P(r) = \{PK,\ sign(inv(PK), pair(H, NPK)),\ inv(PK),\ pair(H, NPK)\}$$

If we now apply the above algorithm to this transition rule, it yields the well-typed composition rules:

- $iknows(H) \cdot iknows(NPK) \Rightarrow iknows(pair(H, NPK))$
- $iknows(inv(PK)) \cdot iknows(pair(H, NPK)) \Rightarrow iknows(sign(inv(PK), pair(H, NPK)))$

This restriction on intruder composition capabilities achieves what we call a *message-bounded AIF* where all rule variables are restricted to instantiation of

a maximum depth. In section 4.4 we show that this bound on substitution is sound.

4.3.2 Intruder Decomposition

For any intruder decomposition rule that extracts subterm(s) from a message, we allow it to use *untyped* variables. This decision does not affect the well-typed AIF model as in a well-typed state, decomposing typed terms can never result in untyped terms. Hence, *untyped* variables could be seen as place holders for possible typed terms.

In addition, we restrict decomposition rules so that a rule is allowed only to specify extraction of subterm(s) of depth at most less-by-one than the original term. In other words, if the term being decomposed has a depth of j, then the decomposition rule can extract subterm(s) of depth at most $j - 1$. Therefore, intruder analysis rules take the form:

$$iknows(f(t_1, \ldots, t_n)) \cdot iknows(s_1) \cdot \ldots \cdot \ldots \cdot iknows(s_n) \Rightarrow iknows(t_{i \in \{1,\ldots,n\}})$$

for some n-ary function symbol $f \in \Sigma$.

For example, the rule

$$iknows(g(x, h(y))) \cdot iknows(x) \Rightarrow iknows(y)$$

for $x, y : untyped$ and $g, h \in \Sigma$ is not allowed according to our restriction.

In fact, this restriction does not affect the expressiveness of the AIF specification language for a large class of protocols. This is true because in most cases, the Dolev-Yao adversary behaviour could be modeled without the need to use such unusual form of intruder analysis. Later on, when showing our soundness result, we will explain the intuition behind this restriction and discuss how it could be removed.

Now, in the light of this modification to the intruder deduction rules, we extend our definition in 4.7 of well-typed transition rules to include the well-typed intruder composition rules.

Definition 4.10 The set of *well-typed transition rules* $R_{T'}$ is the set of well-typed transition rules R_T where untyped intruder composition rules are replaced with a finite set of well-typed intruder composition rules (cf. section 4.3.1) and intruder decomposition rules conform to the restriction in section 4.3.2.

4.4 Soundness

We begin our discussion here by noting that verifying protocol security under a typed model is not a restriction for a large class of protocols. If there is an attack in an untyped model, then there is a similar attack in a typed model - provided that the implementation is reasonable: unambiguous and pairwise disjoint formats [MK14]. This result is due to [HLS03] where bounding is justified by tagging. In fact, such a restriction without loosing attacks is desirable since many type-flaw attacks are unrealistic [HLS03].

Moreover, the desirable and crucial property of our proposed type system for AIF is that the verification of the well-typed model with bounded intruder implies the verification of the well-typed model with unbounded intruder. In other words, if the weakly-typed intruder deduction rules yield an attack, then so does the well-typed attacker model.

Theorem 4.11 *In a typed AIF specification, if there is a reachable attack state using the well-typed rules R_T and unbounded intruder rules, then there is a reachable attack state using the well-typed rules $R_{T'}$ with bounded intruder.*

Proof.

The idea is that under a typed model, arbitrarily complex terms composed by an intruder that do not conform to specific formats do not add to his knowledge and hence does not increase his power in finding attacks.

Consider a typed AIF specification where the attacker is unbounded.

The intruder indefinitely compose and decompose arbitrary messages from his knowledge to produce new ones and he can infinitely continue doing so till he reaches a state S at which there is a derivable *attack* fact. In such an attack trace, there are intermediate states where the intruder may have composed ill-typed messages. Since these malformed messages are rejected by honest agents - due to typing -, they cannot be used to reach the well-typed attack state. This means that there must be a well-typed transition rule (or a set of well-typed transition rules) that take(s) a well-typed state to an attack state; bypassing any intermediate states with ill-typed messages.

Moreover, infinitely composing and decomposing messages do not add useful terms to the intruder knowledge in the typed model. In some trace, let the last term composed by an intruder be ill-typed. Any further composition using this term will yield an ill-typed message too, which does not help an adversary reach

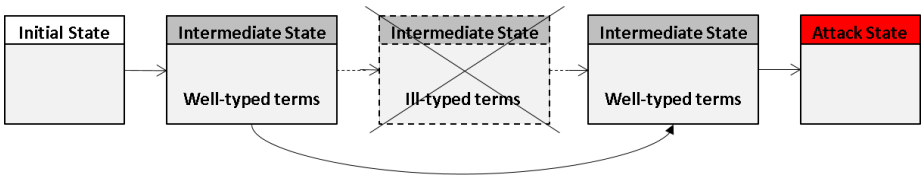

Figure 4.1: Attack trace in a typed model

a well-typed attack state.

Decomposing an ill-typed term can either result in ill-typed terms and/or typed terms. If the resultant term is also ill-typed, then it still unuseful. But, if the decomposition gives a typed term, then it must have already been in the attacker knowledge. In fact, this result is due to the restriction we imposed on the intruder decomposition rules (cf. section 4.3.2). Take for example, the intruder rules:

- $iknows(K) \cdot iknows(h(x)) \Rightarrow iknows(g(k, h(z)))$
- $iknows(g(M_1, h(M_2))) \cdot iknows(M_1) \Rightarrow iknows(M_2)$

Where $g(\cdot,\cdot)$ and $h(\cdot) \in \Sigma$; K is some constant, z is a typed variable and M_1, M_2 are untyped variables. Assume that the message $g(k, h(z))$ is ill-typed (i.e. never occurs in any of the normal protocol interaction steps). The above argument would be invalid if we have allowed such decomposition rule where an intruder can extract a typed term he never knew before (z in our example) from an ill-typed message: $g(k, h(z))$. If such a rule was allowed, then one will not be able to exclude intermediate states with ill-typed messages as indicated in figure 4.1 because it would be possible for the intruder to obtain new terms by decomposing ill-typed messages.

Now, consider a bounded attacker under a typed AIF model.

By bounding the intruder composition to typed terms of maximum depth, the attacker cannot reach any of the ill-typed states of the unbounded intruder discussed above. However, any well-typed reachable state in the unbounded attacker model is also reachable under the bounded attacker. This is because we showed that well-typed states in the unbounded model are reachable through well-typed transition rules and ill-typed states could be skipped without loosing useful knowledge. So the bounded intruder can reach the same set of well-typed states that are reachable by an unbounded adversary.

Therefore, in the typed AIF specification, an attack state that is reachable under unbounded intruder is also also reachable under bounded intruder rules.

4.5 Decidability

The decision problem of the security of protocols is undecidable in general [DLMS99, EG83]. There are three different sources for the undecidability of protocol verification: unbounded intruder messages, number of sessions (executions) and fresh values (constants). In over-approximation techniques like the AIF set-based abstraction, there is no notion of sessions. This is considered an advantage of over-approximation methods over conventional model checking which requires bounded number of sessions. However, in our AIF over-approximation, transition rules of honest agents interactions which create fresh constants - excluding attacker analysis rules - can be applied infinitely by honest participants. This gives a similar notion of unbounded number of sessions in conventional model checking.

Consider the original untyped AIF. Since the specification assumes the unbounded attacker of the Dolev-Yao adversary model, the security of an AIF specification is undecidable, even when no fresh values are used. For example, the attacker can infinitely concatenate messages from his knowledge and produce new ones. [EG83] shows this undecidability result by reduction from the well-known undecidable Post Correspondence Problem (PCP) [EG83]. This makes both the concrete and abstract untyped AIF models undecidable as already noted in [Möd10].

Now, consider a well-typed AIF specification (cf. section 4.2) which limits the intruder deduction capabilities as described in section 4.3 to well-typed terms of maximum depth. In the concrete model, such restriction is not sufficient to yield a decidable verification problem due to the unbounded use of fresh constants. [Möd10] proves the undecidability of the concrete model by showing that Turing machines can be encoded into depth-bounded AIF. Since reachability of states for Turing machines is undecidable, the reachability of attack state is also undecidable in the bounded concrete model.

Before we discuss our decidability result for the bounded abstract AIF model, we need to define an *abstract state*. In the original AIF model, the notion of abstract state is not defined but rather the set of derivable facts from abstract states is over-approximated using Horn clauses.

Definition 4.12 An *abstract state* $abs(S)$ of a state S is the normal state S

(cf. definition 4.6) where all constants values c are replaced with their abstract counterparts $abs_S(c)$ using the abstraction function $abs_S : \mathfrak{A} \rightarrow val(\mathbb{B}^N)$ (cf. section 3.2.1). This definition induces an equivalence class on the set of states where $S_1 \equiv S_2$ if $abs(S_1) = abs(S_2)$ and we denote the equivalence class of a state S by $[S]_{\equiv}$.

We can say that $[S]_{\equiv} \Rightarrow_r^{Abs} [S']_{\equiv}$ if $S \Rightarrow_r S'$ where $\Rightarrow_r^{Abs}$ uses the grounding substitution as specified in definition 3.6 but constants in $\mathfrak{A}$ are replaced with their abstractions.

We now show our decidability result for the well-typed abstracted AIF in the following theorem.

Theorem 4.13 *Let $\overrightarrow{S} = \{S \mid \emptyset \Rightarrow^*_{R_{T'}} S\}$ be the set of reachable states under message-bounded AIF in the concrete model. Then in the abstract message-bounded AIF model, the set $\overrightarrow{S}_{Abs} = \{abs(S) \mid S \in \overrightarrow{S}\}$ of reachable abstract states is finite.*

Proof.

Recall that messages in AIF are terms over the the signature $\Sigma \cup \mathfrak{A}$ and a set of variables $\mathcal{V}$ (cf definition 3.1). The original specification stipulates that the set of variables $\mathcal{V}_{\mathfrak{A}} \subset \mathcal{V}$ can only be instantiated by constants from the countable set $\mathfrak{A}$ of constant "abstractable" symbols.

The first thing to note is that constant symbols of $\mathcal{A}$ are abstracted according to their respective set-membership in the used sets in an AIF specification which yields a set of equivalence classes of abstract symbols (cf. section 3.2.1). Since the number of sets used is finite - say N -, the number of corresponding equivalence classes is also finite: 2^N. This means that the abstraction eliminates the infinite fresh constants from an AIF protocol specification by mapping the infinite set of constant symbols in the concrete model to finitely many equivalence classes in the abstract model. Indeed this means that in the abstract model, variables in $\mathcal{V}_{\mathfrak{A}}$ are substituted by finitely many constants.

The second remark is that in a message bounded AIF specification, both in the concrete and abstract models, intruder deduction rules are restricted to terms of maximum depth (cf. section 4.3.1). This means that rule variables $\mathcal{V}$ are substituted with terms of maximum depth. In other words, the depth restriction of terms guarantees that the closure of intruder knowledge under protocol independent intruder deduction rules is finite.

So in a sense, the abstracted message-bounded AIF model restricts all variables to instantiation of given depth (whether variables of $\mathcal{V}_{\mathfrak{A}}$ or the rest of rule variables $\mathcal{V}$). This means that we have finite number of derivable terms which is another way of saying that we have finitely many reachable abstract states $abs(S)$. It is important here to recall that in the AIF specification, a state (cf. definition 3.4) is a set of facts. In fact, our theorem here states that the set of derivable facts is finite under message-bounded AIF abstraction.)

Worth mentioning is that if we restrict the set of fresh constants to be finite, we can reach a similar decidability result in the concrete message-bounded AIF model as well.

Now, having presented our typing system for the AIF protocol verification by abstraction, we present the implementation of this new typing system for AIF.

Chapter 5

Implementation

The original contribution of the AIF by Mödersheim was accompanied by an implementation of a translator tool. This translator accepts a protocol specification written in a variant of AVISPA IF and translates it into set of first-order Horn clauses as described in chapter 3. The tool supports translation to the syntax of the automatic theorem prover SPASS [WDF+09] and the automatic cryptographic protocol verifier ProVerif [BCAS10].

An important decision that guided our entire implementation steps was to preserve the original AIF translator as is. This meant that we keep all features of the original tool untouched. So one should be able to user the translator with original untyped AIF specifications.

5.1 The Type System

Recall that our proposed type system for AIF introduced in chapter 4 distinguishes between variables of two types only: *atom* and *untyped*. Though *untyped* variables could be used in any rule variables as discussed above, their main purpose is to specify the intruder deduction rules. So we can safely assume that *untyped* variables appear only in the intruder analysis rules.

To declare all other variables to be of *atom* type, we use a new unary fact symbol: $atom(\cdot)$ (it should not occur in any given protocol specification). This predicate symbol is applied to all variables in all specification rules, including the variables of fresh data $\mathcal{V}_{\mathfrak{A}}$. We do this in a transparent way: the user specification written in AIF language is the same but before creating the abstract model, if the user has specified that he wants his specification to be typed, then we insert the *atom* predicate for every variable used on the LHS of every transition rule. We also insert the term $atom(c)$ on the RHS of every rule that creates a new fresh value c.

We remind the reader that the AIF transition rule has the form:

$$LF \cdot S_+ \cdot S_- =[F]\Rightarrow RF \cdot RS$$

where LF and RF are sets of facts, S_+ and RS are sets of positive set conditions, S_- is a set of negative set conditions; and $F \subseteq \mathcal{V}_{\mathfrak{A}}$.

We use the following algorithm to enforce the typing system:

Algorithm 2: Construction of typed AIF specification

Data: Protocol Specification written in the AIF language
Result: Typed AIF protocol specification
begin
  for *each declared enumeration variable V* **do**
    Add the rule: $\lambda V. \Rightarrow atom(V)$;
  for *each normal transition rule (not intruder deduction rules)* **do**
    for *each used variable V and fresh value variable F* **do**
      Add $atom(V)$ to LF;
      Add $atom(F)$ to RF;

5.2 Bounded Intruder

If the user has chosen to translate his protocol specification in the typed AIF model, along with rule typing explained above, we also bound the adversary capabilities as discussed in section 4.3.

For composition rules, we replace all untyped intruder composition rules with bounded rules using algorithm 1. After that, we type those rules using an algorithm similar to 2 that operates on intruder rules - not normal transition rules.

For intruder decomposition rules, these should be the only rules that are permitted to use *untyped* variables. Moreover, we ensure that decomposition is restricted to terms of depth at most 1-less than the original term being decomposed as discussed in section 4.3.2. We carry out this validation through the following algorithm:

Algorithm 3: Validation of intruder decomposition rules for typed AIF

```
Data: Protocol Specification written in the AIF language
begin
    for each rule that uses untyped variable(s) Y do
        if terms(RF) ⊑ terms(LF) and
        Maxdepth(terms(LF)) - Maxdepth(terms(RH)) = 1 then
            Continue;
        else
            Error: Intruder decomposition cannot be used in typed AIF;
```

5.3 Fixedpoint Module

We decided to extend the features of the AIF tool so that it is not just a translator to different verifiers syntax, but also it could be used for verification of the abstract model on its own. To do so, we adopt the fixedpoint computation module from the OFMC tool [BMV05] to our AIF tool. This module works in a similar manner to tools like ProVerif and TA4SP. It computes the smallest set of facts which is closed under each transition rule. Worth mentioning is that since we are verifying protocols in an over-approximated abstract model, reported attacks may be *false positives*. However, when no attacks are found, we can safely say that the concrete protocol is secure because we verified a sound over-approximation of the actual specification.

We do not claim major contribution in this part. Our contribution only integrates the OFMC fixedpoint module into the AIF tool.

5.4 Current Limitations

The original AIF translator tool is very versatile and allows users to specify protocols with great flexibility. The specification language has no special constructs and users define their own enumeration variables, fresh values variables,

sets, public and private function; as well as fact (predicate) symbols. Even the predicate symbols $iknows(\cdot)$ and $attack$ do not have special predefined meanings in AIF. However, the new extensions to the AIF tool assumes the following:

Automatic type enforcement:

- The fact symbol $iknows(\cdot)$ denotes intruder knowledge.
- *untyped* variables should be used only in intruder deduction rules. Normal protocol transition rules should not contain variables of type *untyped*.

Fixedpoint computation:

- The function symbol $pair(\cdot, \cdot)$ denotes composition by concatenation. Any message concatenation operation should use $pair(\cdot, \cdot)$.

It is to be noted that these limitations are due to the scope and time limit of the project. They do not have any impact on the soundness of the typed model. In fact, one can opt to write the typed specification manually following the same ideas from our algorithms. We plan to remove these implementation restrictions in future work.

We implemented the mentioned algorithms on top of the original AIF translator. The AIF tool now can:

- Generate Horn clauses representation of the abstract model both untyped and typed in the syntax of SPASS and ProVerif.
- Use a built-in fixedpoint computation module to verify the abstracted protocol model both typed and untyped.

In the next section we discuss how did we experiment the different options of the new AIF tool on some examples.

Chapter 6

Experimental Results

We tested our modifications on the AIF tool using the set of protocol examples originally provided with the translator [Möd10]. We wrote manual specification for the typed model of some protocols where applicable: see section 5.4.

Our main contribution is the typing system for AIF. therefore, the major part of our experiments was focused on testing untyped versus typed protocol specification. It is unfortunate that we could not dedicate much time to experiments on the new fixedpoint module. However, this is planned and already in progress for future work.

6.1 Results of the Typing System

During our experiments, we decided to focus on the automatic protocol checker ProVerif since our initial trials with SPASS showed no major performance difference between typed and untyped model.

Checking typed models against untyped models of the protocol examples from the original library provided with the AIF tool yielded the following results:

Protocol	Agents	ProVerif (untyped)	ProVerif (typed)	Result
Key-server	a, i, s	0 s	0 s	Verified
	s, b, c, i, s	0 s	0 s	Verified
TLS (simplified)	a, i	7 s	8 s	Verified
	a, b, i	0 s	0 s	Verified
NSPK	a, b, i	0 s	0 s	Attack
NSL	a, b, i	0 s	0 s	Verified

Though these results are not very interesting since the untyped model already terminates in no time, they give us an idea about the correctness of our typed model

However, we obtained more interesting results from two other protocols:

- A system for secure vehicle communication from SEVECOM project [LBH+06].

SEVECOM is a system designed for securing communication with cars for the purposes of confidentiality, authentication and accountability. Cars are equipped with tamper-proof hardware security module (HSM) to store necessary keys required for encryption, decryption and verification. There is a certificate authority (CA) with which HSM keys are registered.

The specification of part of SEVECOM is provided with AIF library and it is explained in details in [MM11]. We picked the trivial scenario where the system uses two private keys of the HSM and both are leaked to the attacker. The original untyped model reports the trivial attack only on SPASS, whereas ProVerif times out. Yet, under the typed model, ProVerif was able to find the attack in 3 minutes for 1 HSM. The verification time increases very much with 2 HSM. This is understood because for a model with N sets, the number of equivalence classes in the abstract model is 2^N.

- The contract signing protocol based on Fair Exchange: Asokan Shoup Waidner (ASW) [ASW00].

This protocol is designed for the purpose of fairly signing a contract between two parties: either they both get a valid signed contract, or both don not get it. To achieve this goal, a trusted third party (TTP) must be involved to resolve disputes on contracts. TTP keeps track of which contracts are valid and which are invalid by maintaining a database of contracts.

We experiment our typing system on the specification of the ASW protocol provided in the library of the original AIF tool. Unfortunately, the typed model times out on both ProVerif and SPASS though the untyped model terminates on both tools. This is currently under investigation. This seems to be attributed to the specification of the ASW protocol as it uses *untyped* variables in normal state transition rules (see the specification for more details).

Chapter 7

Conclusion

Automatic verification of security protocols is an active area of research. There are two main models for verifying security protocols: the computational model [GM84] and the symbolic model [NS78, DY83]. Protocol verification by over-approximation and abstraction to set of Horn clauses is a very interesting and promising technique for proving security of protocols in the symbolic model. The main advantages of over-approximation techniques are protocol verification with unbounded number of sessions; on the contrary to classic model checking which requires bounded number of sessions and bounded intruder. Also, proving an over-approximation is attack-free implies the safety of the concrete protocol.

The novel AIF set-based abstraction technique extends the scope of verification by traditional over-approximation to complex protocols and web services that maintain databases of keys or status of different subjects (access rights for example). Traditional approaches cannot handle such systems due to the monotonicity of deduction in contrast with non-monotonicity of facts about these complex systems because revocation of keys - for example - means that the set of true facts about the system is smaller. AIF abstractions by set membership handle this situation in a smart by updating the abstraction, when necessary, through a new type of rule called term implication rule. The term implication rule could be encoded in Horn clauses representation. This method was used to verify different protocols and it has proven successful [Möd10].

Since protocol verification is undecidable in general, the AIF abstraction is no exception. Termination is not guaranteed and experiments already shows this. That was the motivation behind our work. We extended the AIF abstraction by set membership with a type system. We use this type system to bound the intruder capabilities so that an attacker can formulate messages of a maximum size. Our theoretical contribution is the typing system for AIF. We also proved the soundness and decidability of the new typed abstracted AIF model. We showed how we use this type system to bound the intruder deduction rules of the Dolev-Yao adversary model. On the practical side, we extended the AIF translator tool so that it can translate an untyped AIF specification to typed one before encoding the abstract model into Horn clauses. This gives us a message-bounded over-approximated representation of the protocol. We can use tools like SPASS and ProVerif to check the resultant abstract protocol model. Comparing the verification time of typed AIF models versus the untyped one for some example shows promising results. One of the modeled SEVECOM scenarios that timeout in the untyped model did terminate in few minutes using our typed model.

Additionally, we extended the capabilities of the AIF tool so that it becomes a stand-alone verifier; beside being a translator to different back-end tools. We borrowed the novel fixedpoint computation module from the OFMC tool. Our contribution is integrating this module into the AIF tool so that users can - optionally - verify their protocols without the need to use other external tools.

7.1 Future Work

Having obtained promising results, we believe there are still many directions to improve and extend our work. First, the restriction we enforced on intruder decomposition as discussed in section 4.3.2 could be removed by extending the sett of message patterns an intruder may ever know in a specific way. Second, we plan to remove the implementation restrictions presented in section 5.4. We foresee doing so in different way. For instance, one can enforce distinction between intruder deduction rules and normal protocol rules by using different specifiers for section names in the AIF specification files. Also, enriching the syntax of the AIF language by allowing explicit types is very appealing, specially for the purpose of verifying well-typed protocols models. Another important expansion to our work is to add more protocol examples to the library of AIF. This will give us more insights on how rigorous our results is both in typed and untyped models.

Finally, an important direction for us is to ensure that the integration of the

fixedpoint module is independent from our implementation specific decisions. For example, it currently depends on the $pair(\cdot, \cdot)$ function symbol to preform composition and decomposition of terms. This makes it unable to verify protocols that uses different symbols for concatenation of messages. Many of these different direction suggest that it might be handy to incorporate and use a higher level languages for protocol specification - since AIF is a bit low-level. We think that it is not very straightforward for normal user to be able to write their own specification in the AIF language.

Bibliography

[ABB+05] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification*, pages 281–285. Springer, 2005.

[ASW00] Nadarajah Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *Selected Areas in Communications, IEEE Journal on*, 18(4):593–610, 2000.

[BCAS10] Bruno Blanchet, V Cheval, X Allamigeon, and B Smyth. Proverif: Cryptographic protocol verifier in the formal model, 2010.

[BCM11] David Basin, Cas Cremers, and Catherine Meadows. Model checking security protocols. *Handbook of Model Checking*, 2011.

[Bla08] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *Dependable and Secure Computing, IEEE Transactions on*, 5(4):193–207, 2008.

[BMV05] David Basin, Sebastian Mödersheim, and Luca Vigano. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[Boi] Y Boichut. Tree automata for security protocols (ta4sp) tool.

[BP03] Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *Foundations of Software Science and Computation Structures*, pages 136–152. Springer, 2003.

[DLMS99] Nancy A Durgin, Patrick D Lincoln, John C Mitchell, and Andre Scedrov. Undecidability of bounded security protocols. In *In Workshop on Formal Methods and Security Protocols (FMSP??? 99*, 1999.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[DY83] Danny Dolev and Andrew C Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[EG83] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 34–39. IEEE, 1983.

[EMM09] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-npa: Cryptographic protocol analysis modulo equational properties. In *Foundations of Security Analysis and Design V*, pages 1–50. Springer, 2009.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[HLS03] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. *Journal of Computer Security*, 11(2):217–244, 2003.

[LBH+06] Tim Leinmüller, Levente Buttyan, Jean-Pierre Hubaux, Frank Kargl, Rainer Kroh, Panagiotis Papadimitratos, Maxim Raya, and Elmar Schoch. Sevecom-secure vehicle communication. In *IST Mobile and Wireless Communication Summit*, number LCA-POSTER-2008-005, 2006.

[Low95] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, November 1995.

[MK14] Sebastian Modersheim and Georgios Katsoris. A sound abstraction of the parsing problem. In *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*, pages 259–273. IEEE, 2014.

[MM11] Sebastian Mödersheim and Paolo Modesti. Verifying sevecom using set-based abstraction. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International*, pages 1164–1169. IEEE, 2011.

[Möd10] Sebastian Alexander Mödersheim. Abstraction by set-membership: verifying security protocols and web services with databases. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 351–360. ACM, 2010.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978.

[Pau94] Lawrence C Paulson. *Isabelle: A generic theorem prover*, volume 828. Springer Science & Business Media, 1994.

[RT01] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with finite number of sessions is np-complete. 2001.

[SS98] Vitaly Shmatikov and Ulrich Stern. Efficient finite-state analysis for large security protocols. In *Computer Security Foundations Workshop, 1998. Proceedings. 11th IEEE*, pages 106–115. IEEE, 1998.

[WDF+09] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. Spass version 3.5. In *Automated Deduction–CADE-22*, pages 140–145. Springer, 2009.