A Spatial-Epistemic Logic for Reasoning about Security Protocols

Bernardo Toninho
CITI and Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Carnegie Mellon University, Pittsburgh PA, USA
Btoninho@cs.cmu.edu

Luís Caires
CITI and Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
Luis.Caires@fct.unl.pt

Reasoning about security properties involves reasoning about *where* the information of a system is located, and how it evolves over time. While most security analysis techniques need to cope with some notions of information locality and knowledge propagation, usually they do not provide a general language for expressing arbitrary properties involving local knowledge and knowledge transfer. Building on this observation, we introduce a framework for security protocol analysis based on dynamic spatial logic specifications. Our computational model is a variant of existing π-calculi, while specifications are expressed in a dynamic spatial logic extended with an epistemic operator. We present the syntax and semantics of the model and logic, and discuss the expressiveness of the approach, showing it complete for passive attackers. We also prove that generic Dolev-Yao attackers may be mechanically determined for any deterministic finite protocol, and discuss how this result may be used to reason about security properties of open systems. We also present a model-checking algorithm for our logic, which has been implemented as an extension to the SLMC system.

1 Introduction

Among the several artifacts in the field of computer security, security protocols are indubitably a fundamental subject of study and research [12, 11]. Security protocols serve a variety of purposes, ranging from secrecy and authentication to forward secrecy and deniable encryption. A common trait of these protocols is their notoriously difficult design, which often leads to unforeseen vulnerabilities.

Therefore, it becomes essential to develop techniques that ensure the correctness of protocols, with respect to some specification of the properties they aim to establish. A wide range of language-based techniques have been proposed to analyze protocols and their correctness, such as type systems, process calculi or static analysis [3, 6, 2] which in many cases result in successful tools [5, 4, 13, 9].

In this paper we propose a framework for protocol analysis based on process calculus models and logic specifications. While the usage of process calculi and logic in this context is not new [14, 8, 2], our approach stems from the fact that many interesting properties of such systems are often a function of what information the several parts of a system may or may not obtain. While other frameworks (e.g., Avispa [4] and Casper [13]) allow one to efficiently verify a wide range of interesting security properties, these are not usually stated in this high-level knowledge oriented approach.

Our contribution consists of a dynamic spatial epistemic logic that allows reasoning about systems (modelled in a variant of the applied π-calculus [2]) at three levels: the *dynamics* of systems and subsystems, the *spatial* arrangement of systems and subsystems, and the *knowledge* (the obtainable information)

K. Chatzikokolakis and V. Cortier (Ed.): 8th International Workshop
on Security Issues in Concurrency (SecCo 2010)
EPTCS 51, 2011, pp. 1–15, doi:10.4204/EPTCS.51.1

```
S(pkp,pkq, sks, pks) = c?(h).select { [pkp = getpk(h)].c!(enc_as(pkp,sks)).S(pkp,pkq) ;
                                      [pkq = getpk(h)].c!(enc_as(pkq, sks)).S(pkp,pkq) };

defproc P(skp,hostQ,pks) = c!(hostQ).c?(m).let pkQ = dec_as(m,pks) in
                            new sK in c!(enc_as(sK,pkQ)).c!(enc(v,sK)).ok!(v);

Q(skq) = c?(m1).let sK = dec_as(m1,skq) in c?(m2).let val = dec(m2,sK) in ok!(val);

defproc Sys = new skp, skq in let pkp = pk(skp) in let pkq = pk(skq)
                           in let hP = host(pkp) in let hQ = host(pkq)
                           in (S(pkp,pkq) | P(skp,hQ) | Q(skq));

World = Sys | Attacker(Sys);

prop pqK = eventually (knows v | knows v | not (knows v))
           and always (2 | not (knows v));

check World |= pqK;

* Process World satisfies the formula pqK *
```

Figure 1: A Motivating Example

of systems and subsystems. The goal is to produce an expressive property language with which we can reason about a protocol by separating it into its different agents (malicious and otherwise), and then reason about the knowledge they can obtain and how it can evolve over time. This enables us to express interesting security properties in a very direct way (eg. agents P and Q can obtain value v, while agents A and S cannot). To clarify our approach, consider the example of Fig. 1.

We have a system Sys composed of three processes: P, Q and a key distribution server S. P wishes to exchange a value v with Q. To do so, he requests Q's public key from S, which S emits in a signed message. P then uses it to encrypt a generated symmetric session key and sends the key to Q. Afterwards P will send v encrypted with the session key and terminate. Q will receive the message, decrypt it to obtain v and terminate. We further model the system running with a malicious agent, defined through the primitive `Attacker(Sys)`. This agent consists of a Dolev-Yao attacker which we discuss in Section 4.1.

For this protocol to be correct, it must be the case that the malicious agent interacting with the system can never know v. Consider, however, a slightly stronger property: P and Q want to exchange v securely (with respect to the malicious agent) but they also do not completely trust the server S. They trust it to at least distribute the appropriate keys, but want some assurances that even though S operates according to protocol, it doesn't obtain the value v by observing the data exchanges between P and Q.

This property, while not impossible to state in other frameworks, would usually require some sort of *ad-hoc* modification to the model (e.g, internalizing the server in an attacker, which seems like an indirect strategy at best and may not necessarily yield the correct model). In our framework, the property can be directly stated by combining our epistemic and spatial operators. A formula that reflects such a property is pqK: first we state that the system can evolve to a configuration where two of its subsystems (P and Q) know v, but the remaining parts of the system do not. This illustrates the expressiveness of the logic in terms of reasoning about the knowledge of several parts of the complex system. Secondly, we state that throughout *all* executions of the system, a part of it will never know v (2 indicates that there must be two agents running with the part that does not know v – a precise definition is given in Section 3). By combining spatial reasoning with epistemic reasoning, we can state rich properties of the knowledge of agents (and groups of agents) – both adversaries and principals – within a complex system, and how they can share or restrict that knowledge over time.

While our framework is aimed at reasoning about closed systems, meaningful analyses of security protocols must necessarily consider attackers. Traditionally, attackers are modelled by an adversarial environment which interacts with the protocol. In our closed system approach, we develop a way internalizing an arbitrary attacker within a closed system by *automatically* (or semi-automatically) deriving a process representation of such an attacker. This representation makes use of special primitives built into the process calculus to greatly simplify the actual modelling of the attacker, to the extent that even if the attacker generation is done semi-automatically, it *never* requires us to actually encode specific attacks. In this work, we show that it is possible to *automatically* derive an attacker (behaving as a Dolev-Yao adversarial environment) for any finite protocol. To fully automate our technique (at the implementation level), further work is needed (as discussed in Section 4.1); the focus of the current paper being essentially on the expressiveness issues. In any case, we already provide tool support for arbitrary passive attackers and for *bounded* Dolev-Yao attackers (where the bound concerns the size of generated messages); this technique can already be used to automatically find attacks, eg., as illustrated in the example of Section 4.2.

The technical contributions of this work are as follows. We develop a process calculus model for security protocols (Section 2.2), inspired in existing π-calculi, supporting explicit modeling of adversarial agents, at an adequate level of abstraction. We introduce a new dynamic spatial epistemic logic (Section 3), oriented for reasoning about spatial distribution of information. We develop a logic-based theory of knowledge deduction (Section 3.2) for our models, proved sound, complete and decidable. This presentation was used in our model-checking algorithms. We discuss attacker representations (Section 4.1), and how it is possible to produce a generic Dolev-Yao attacker for finite protocols. We also show how to model and verify correspondence assertions (Section 4.2) in our framework. Finally, we implemented a model-checking algorithm for the logic as an extension to the SLMC tool, producing the first proof of concept tool aimed at security protocol analysis using spatial logic model checking. The proofs of our technical results are detailed in [17].

2 Process Model

In this section we introduce our process model, starting with some preliminary notions on terms and equational theory and then introducing our process calculus.

2.1 Terms and Equational Theories

Data exchanged by processes is modeled by terms of a term algebra. In order to capture cryptographic operations and data structuring, we will consider term algebras with equational theories (cf. [2]).

We assume an infinite set of variables ranged over by x, y, z, an infinite set of names Λ ranged over by m, n and range over terms with s, t, v. Terms are defined from names and variables by applying function symbols. We thus consider a given term algebra to be defined from a signature Σ and an equational theory E that defines the "semantics" of the function symbols in Σ. An equational theory is a congruence relation defined by a set of equations of form $t = s$.

In certain circumstances, an equational theory may give rise to a set of rewrite rules by orienting each equation to produce the rule $t \to s$, in such a way that two terms are equal modulo E whenever that have a common reduct under rewriting. This is the case of subterm convergent equational theories [1], which are the ones that we will focus on in this work (other equational theories, such as AC theories, can also be applied in this fashion, however with a slightly different formal treatment as detailed in [1]). A subterm convergent system is a convergent rewrite system in which in every rewrite rule the right-hand side is a proper subterm of the left-hand side. In this paper, we will assume a general rewrite theory $\mathcal{R}$ subject to the conditions above. Given a rewrite rule $t \to s$, we call the outermost function symbol in t a *destructor*, since the application of the rule may open the internal structure of inner terms in t to produce the term s. We classify the remaining function symbols, that never occur as a destructor, as *constructors*. For example, for signature $\Sigma \triangleq \{\mathtt{enc}/2;\mathtt{dec}/2\}$ and equational theory $E \triangleq \{\mathtt{dec}(\mathtt{enc}(x,y),y) = x\}$, `dec` is a destructor and `enc` a constructor. We range over constructors with f and destructors with δ.

We denote the set of names of a term T by $names(T)$ and the depth of a term as $|T|$ (the depth is the length of the longest path in the tree representation of the term). We state that a term is ground if it does not contain variables. We denote by $=_E$ the usual congruence relation induced by the set of equations E (which can be decided through term rewriting since R is convergent). We write $\mathfrak{F}(\psi)$ for the DY (Dolev-Yao) equational closure of a set of terms ψ, that is, the set of all values (destructor-free terms) generated by terms of ψ through function application, modulo the equational theory. This closure represents all possible information that may be produced from a set of terms while following the rules of the equational theory, which if we interpret a set of terms as a set of messages, is the usual notion of knowledge from the Dolev-Yao model.

Definition 2.1 (Equational Closure) *Given a rewrite theory $\mathcal{R}$, the DY equational closure of a set of terms ψ, noted $\mathfrak{F}(\psi)$, is the least set of terms such that:*

1. $\psi \subseteq \mathfrak{F}(\psi)$
2. $\forall f \in \Sigma.$ *if f a constructor and* $t_1,\ldots,t_k \in \mathfrak{F}(\psi)$ *then* $f(t_1,\ldots,t_k) \in \mathfrak{F}(\psi)$
3. $\forall \delta \in \Sigma.$ *if δ a destructor and* $t_1,\ldots,t_k \in \mathfrak{F}(\psi)$ *and* $\delta(t_1,\ldots,t_k) \to t'$ *then* $t' \in \mathfrak{F}(\psi)$

When interpreting the DY equational closure of a set of terms as obtainable knowledge, we can state knowledge derivation through term derivation.

Definition 2.2 ***(Knowledge Derivation).*** *Given sets of terms ψ and ϕ, we say that ϕ may be derived from ψ (written $\psi \models \phi$) if and only if $\phi \subseteq \mathfrak{F}(\psi)$.*

The general idea is that one may can derive a piece of information if it can be generated by combining pieces of information using the rules of the equational theory. Given these basic notions relative to terms, equational theories, and knowledge derivation, we may now present our process calculus model.

2.2 Process Calculus

It is known that the high level of abstraction of the π-calculus, convenient from a foundational perspective, is not suitable for modeling cryptographic techniques as necessary for analyzing security protocols.

We therefore adopt an extension to the π-calculus that extends the base values of the language with functional terms (cf. Section 2.1), that can be seen as a fragment of the Applied π-calculus [2]. We choose this calculus over the applied π-calculus mainly for simplicity reasons, not requiring active substitutions nor frames given that our goal is to use our logic to observe terms.

We model cryptographic operations by defining such operations in a term algebra. The calculus is thus aimed at the explicit modeling of agents involved in security protocols, both principals and adversaries. Principals are modeled standardly, using terms to model cryptographic terms. Adversaries are modeled as processes (cf. Section 4.1) using the attacker output prefix - a non-deterministic output of terms that can be generated from known values, which enables reasoning directly about attacker knowledge using our logic.

Definition 2.3 (Processes) *Given a signature Σ, an infinite set of names ranged over by m, n, and an infinite set of variables ranged over by x, y, z, the set of processes (P, Q), of actions α and of terms T are defined in Fig. 2.*

$$
\begin{array}{rcll}
P,Q & ::= & \mathbf{0} & \text{(Null Process)} \\
 & | & P \mid Q & \text{(Parallel Composition)} \\
 & | & (\nu n)P & \text{(Name Restriction)} \\
 & | & \alpha.P & \text{(Action Prefix)} \\
 & | & P + Q & \text{(Choice)} \\
 & | & \text{let } x = T \text{ in } P & \text{(Let Construct)}
\end{array}
\qquad
\begin{array}{rcll}
\alpha & ::= & m(x) & \text{(Input)} \\
 & | & m\langle T\rangle & \text{(Output)} \\
 & | & m\langle * \rangle & \text{(Attacker Output)} \\
 & | & [T_1 = T_2] & \text{(Test)} \\
 & & & \\
T & ::= & n & \text{(Name)} \\
 & | & x & \text{(Variable)} \\
 & | & f(T_1,\ldots,T_a) & \text{(Function)}
\end{array}
$$

Figure 2: Process Calculus Syntax

$$
sub(\delta(t_1,\ldots,t_n)) \triangleq sub(t_1) \cup \cdots \cup sub(t_n) \qquad \frac{n \textit{ is a name}}{sub(n) \triangleq n} \qquad \frac{x \textit{ is a variable}}{sub(x) \triangleq \emptyset}
$$

$$
\frac{\nexists t_i \textit{ with a variable or a destructor}}{sub(f(t_1,\ldots,t_n)) \triangleq f(t_1,\ldots,t_n)} \qquad \frac{\exists t_i \textit{ with a variable or a destructor}}{sub(f(t_1,\ldots,t_n)) \triangleq sub(t_1) \cup \cdots \cup sub(t_n)}
$$

Figure 3: Relevant Subterms

Before introducing the semantics of our calculus, we present some definitions that pertain to obtaining the *relevant terms* of a process that are necessary for our semantics.

A destructor function symbol denotes computation at the term level. If such computations are valid (under the equational theory), then the term containing the destructor can be rewritten as one that only has constructors. On the other hand, if such a term cannot be reduced (e.g $\mathtt{dec}(\mathtt{enc}(m,k_1),k_2)$), it has no interesting meaning and has no place being communicated. To obtain the values (destructor free normal forms) of a process, we define a relation $\vdash_k$ that extracts the set of values ψ that occur in a process ($P \vdash_k \psi$). However, some care is needed in the definition of $\vdash_k$ since a term may contain bound names or variables. For instance, in the process $a(x).a\langle \mathtt{dec}(x,k)\rangle.\mathbf{0}$, the term $\mathtt{dec}(x,k)$ is not a proper value since it contains the variable x. In these situations, our extraction has to be such that it will produce a set containing k but not x (nor $\mathtt{dec}(x,k)$). Similarly, when we consider the terms that are to be the object of our attacker output, while it is true that outputting a term containing a variable would be senseless, it is correct to output a term that contains a restricted name, even though the attacker may not be able to use the name in other messages.

To take all this into account, we define a procedure *sub* that extracts the *relevant subterms* (not containing variables or destructors) of a term, and a procedure $\uparrow$ used to eliminate terms with restricted names.

Definition 2.4 (Relevant Subterms) *Given a term M we define the set of its relevant subterms, written $sub(M)$, by the rules of Fig. 3.*

Definition 2.5 (Name Occurrence Term Removal) *We define the removal of terms from a set ψ in which the name x occurs, $\psi \uparrow x$, as:* $\psi \uparrow x \triangleq \{t \mid t \in \psi : x \notin names(t)\}$.

Definition 2.6 (Relevant Term Extraction) *Given a process P, the set ψ of relevant terms of P, written $P \vdash_k \psi$, is defined by the rules of Fig. 4.*

For our attacker output we collect all ground terms that occur in the process, which we denote by $gt(P)$.

The semantics of our calculus are defined standardly, modulo α-conversion of bound names and variables, by a structural congruence relation, labelled transition and reduction, as follows. We denote by $fn(P)$ and $fv(P)$ the set of free-names and free-variables of process P, respectively.

$$\frac{P \vdash_k \varphi \quad Q \vdash_k \psi}{P + Q \vdash_k (\varphi \cup \psi)} \qquad \frac{P \vdash_k \varphi \quad Q \vdash_k \psi}{P \mid Q \vdash_k (\varphi \cup \psi)} \qquad \frac{P \vdash_k \varphi}{n(x).P \vdash_k \varphi} \qquad \frac{P\{n \leftarrow M\} \vdash_k \varphi}{\text{let } n = M \text{ in } P \vdash_k \varphi \cup sub(M)}$$

$$\frac{}{\mathbf{0} \vdash_k \emptyset} \qquad \frac{P \vdash_k \varphi}{x\langle M\rangle.P \vdash_k \varphi \cup sub(M)} \qquad \frac{P \vdash_k \varphi}{(\nu n)P \vdash_k \varphi \uparrow n} \qquad \frac{P \vdash_k \varphi}{[M = N].P \vdash_k \varphi \cup sub(M) \cup sub(N)}$$

Figure 4: Relevant Term Extraction

$$\begin{array}{ll}
n \notin fn(P) \cup fv(P) \Rightarrow P \mid (\nu n)Q \equiv (\nu n)(P \mid Q) & P \mid \mathbf{0} \equiv P \\
(\nu n)\mathbf{0} \equiv \mathbf{0} & P \mid Q \equiv Q \mid P \\
(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P & P \mid (Q \mid R) \equiv (P \mid Q) \mid R \\
M =_E M' \Rightarrow \text{let } x = M \text{ in } P \equiv \text{let } x = M' \text{ in } P & P + Q \equiv Q + P \\
M =_E M' \Rightarrow m\langle M\rangle.P \equiv m\langle M'\rangle.P & P + (Q + R) \equiv (P + Q) + R \\
M_1 =_E M_1' \Rightarrow [M_1 = M_2].P \equiv [M_1' = M_2].P & [M_1 = M_2].P \equiv [M_2 = M_1].P
\end{array}$$

Figure 5: Structural Congruence

Definition 2.7 (Structural Congruence) *Structural congruence $\equiv$ is the least congruence relation on processes defined by the rules of Fig. 5.*

We augment the standard structural congruence laws of the π-calculus with rules that equate processes modulo the equality $=_E$ of the equational theory. These laws are essential in our semantics because they allow us to block processes performing actions that use terms that are not values (i.e. terms that contain destructors).

Our semantics, which we now present, capture these *destructor freedom* conditions. If a process is attempting to use a term that contains a destructor, we use structural congruence to rewrite the term destructor-free and reduction proceeds. If the term cannot be rewritten destructor-free, reduction halts. These restrictions ensure that all received terms are actual values, and not some arbitrary erroneous term. Note the semantics of our attacker output, expressed in the *Attacker* rule, that enable the output to emit any message that can be generated by the process, given its ground terms and some fresh values.

Definition 2.8 (Reduction Semantics) *The reduction relation $P \longrightarrow Q$ over closed processes is defined as the least relation closed under the rules of Fig. 6.*

Definition 2.9 (Labelled Transition Semantics) *The labelled transition relation $P \xrightarrow{\alpha} Q$ is the least relation on closed processes closed under the rules of Fig. 7.*

Our labelled semantics is not intended to characterize a complete notion of behavioral equivalence as could be expected, but rather to allow the observation of actions in our logic. Despite not belonging to the scope of this work, we can point out that our labelled semantics do not allow for a complete characterization of behavioral equivalence, in the sense that our rules reveal information in a way that induces a higher discriminative power then that of behavioral equivalence.

3 Logic

Considering it is common to reason about security by reasoning about the knowledge of principals, we explore key aspects of dynamic spatial logics, such as local reasoning, to develop a logic that can reason about epistemic, dynamic and spatial properties of agents.

We propose an extension to a dynamic spatial logic [7] to enable reasoning at the term level. Our extension consists of adding two epistemic modalities: $\mathbb{K}\phi$ denotes the ability of an agent to derive ϕ from its knowledge, and $\mathsf{S}x.A$ allows us to mention values that are only known by an agent (e.g. secrets).

$$\frac{M \textit{ is destructor-free}}{let\ x = M\ in\ P \longrightarrow P\{x \leftarrow M\}}\textit{(Let)} \qquad \frac{M \textit{ is destructor-free}}{n\langle M\rangle.P + R \mid n(x).Q + S \longrightarrow P \mid Q\{x \leftarrow M\}}\textit{(Sync)}$$

$$\frac{M_1 \textit{ and } M_2 \textit{ are destructor-free} \quad M_1 =_E M_2}{[M_1 = M_2].P \longrightarrow P}\textit{(Test)} \qquad \frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}\textit{(Par)} \qquad \frac{P \longrightarrow Q}{(\nu n)P \longrightarrow (\nu n)Q}\textit{(Scope)}$$

$$\frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}\textit{(Cong)} \qquad \frac{M \in \mathfrak{F}(ct(Q) \cup \overline{n}) \quad \overline{n} \textit{ fresh}}{c(x).P + R \mid c\langle * \rangle.Q + S \longrightarrow (\nu\overline{n})(P\{x \leftarrow M\} \mid Q)}\textit{(Attacker)}$$

Figure 6: Reduction Semantics

$$\frac{P \longrightarrow Q}{P \xrightarrow{\tau} Q}\text{ (Tau)} \qquad \frac{M \textit{ is destructor-free}}{n\langle M\rangle.P \xrightarrow{n\langle M\rangle} P}\text{ (Out)} \qquad \frac{M \textit{ is destructor-free}}{n(x).P \xrightarrow{n(M)} P}\text{ (Inp)}$$

$$\frac{M \in \mathfrak{F}(gt(P) \cup \overline{s}) \quad \overline{s} \textit{ fresh}}{n\langle * \rangle.P \xrightarrow{\nu\overline{s}.n\langle M\rangle} P}\text{ (AttackerOut)} \qquad \frac{P \xrightarrow{\alpha} Q \qquad \forall n \in \overline{u}\text{: n} \notin names(\alpha)}{(\nu\overline{u})P \xrightarrow{\alpha} (\nu\overline{u})Q}\text{ (Res)}$$

$$\frac{P \xrightarrow{n\langle M\rangle} P' \quad \overline{s} \subseteq names(M) \text{ and } \overline{s} \subseteq \overline{u} \quad \overline{u'} = \overline{u} \setminus \overline{s}}{(\nu\overline{u})P \xrightarrow{\nu\overline{s}.n\langle M\rangle} (\nu\overline{u'})P'}\ (\textit{BoundOut}) \qquad \frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q}\ (\textit{Cong})$$

Figure 7: Labelled Transition Semantics

Our intent is to couple the ability to reason about properties of space and behavior with that of reasoning about derivable information modulo the equational theory. Our notion of knowledge is therefore the ability of an agent to derive terms from the information it possesses.

3.1 Syntax and Semantics

The syntax and semantics of our logic are presented in Fig. 8. We refer to ϕ, ψ as knowledge formulas and ambivalently use ϕ, ψ to denote both knowledge formulas and finite sets of terms. The boolean connectives are standard. $\mathbf{0}$ denotes the empty process; $A \mid B$ denotes a process that can be partitioned in two components, one satisfying A and the other satisfying B; $\mathsf{H}x.A$ allows us to mention restricted names of processes in formulas; $\alpha.A$ denotes a process can perform action α and continue as a process satisfying A; $\Box A$ and $\Diamond A$ denote "always in the future" and "sometime in the future", respectively. $\mathbb{K}\phi$ holds of a process that has the ability to derive the terms denoted by ϕ, that is, the ability to know ϕ; $\mathsf{S}x.A$ holds of a process that satisfies property A that depends on a value that is secret to a process – a term containing a restricted name. It is also useful to define an auxiliary *counting predicate* (written as $\mathbf{n}$, where n is a natural number), that allows us to count the number of sub-processes within a process. For instance, a process consisting of a single thread would satisfy the formula $\mathbf{1}$ defined as $\neg\mathbf{0} \wedge \neg(\neg\mathbf{0} \mid \neg\mathbf{0})$, while a process consisting of two sub-processes would satisfy the formula $\mathbf{2}$ defined as $\neg\mathbf{0} \wedge \neg\mathbf{1} \wedge \neg(\neg\mathbf{0} \mid \neg\mathbf{0} \mid \neg\mathbf{0})$, and so on.

With this logic, we can state properties about the knowledge of agents (and not only adversarial ones) over time, such as "it is never the case that the secret key is known by 3 subsystems":

$$\neg\Diamond\mathsf{H}\ key.(\mathbb{K}\ key \mid \mathbb{K}\ key \mid \mathbb{K}\ key)$$

or "it is always the case that 2 agents know the key and one does not": $\Box\mathsf{H}\ key.(\mathbb{K}\ key \mid \mathbb{K}\ key \mid \neg\mathbb{K}\ key)$. Since the semantics of our logic blur together processes that are structurally congruent (e.g. $P \mid Q$ and

$$
\begin{array}{rcll}
A,B & ::= & \mathbf{T} & \text{(True)} \\
& | & \neg A & \text{(Negation)} \\
& | & A \wedge B & \text{(Conjunction)} \\
& | & \mathbf{0} & \text{(Void)} \\
& | & A \mid B & \text{(Composition)} \\
& | & \mathsf{H}x.A & \text{(Hidden quantification)} \\
& | & \alpha.A & \text{(Action)} \\
& | & \Box A & \text{(Always)} \\
& | & \Diamond A & \text{(Eventually)} \\
& | & @n & \text{(Free-name Predicate)} \\
& | & \mathbb{K}\varphi & \text{(Knowledge)} \\
& | & \mathsf{S}x.A & \text{(Secret quantification)} \\
\\
\phi,\psi & ::= & \varphi \wedge \psi & \text{(Conjunction)} \\
& | & t & \text{(Term)} \\
& | & \top & \text{(True)}
\end{array}
$$

$$
\begin{array}{lcl}
P \models \mathbf{T} & \triangleq & \mathit{True} \\
P \models \neg A & \triangleq & \mathit{not}\ P \models A \\
P \models A \wedge B & \triangleq & P \models A\ \mathit{and}\ P \models B \\
P \models \mathbf{0} & \triangleq & P \equiv \mathbf{0} \\
P \models A \mid B & \triangleq & \exists Q,R.P \equiv Q \mid R\ \mathit{and}\ Q \models A\ \mathit{and}\ R \models A \\
P \models \mathsf{H}x.A & \triangleq & \exists Q.P \equiv (\nu n)Q\ \mathit{and}\ Q \models A\{x \leftarrow n\} \\
P \models \alpha.A & \triangleq & \exists Q.P \xrightarrow{\alpha} Q\ \mathit{and}\ Q \models A \\
P \models \Box A & \triangleq & \forall Q\ \text{s.t}\ P \xrightarrow{\tau^*} Q\ \mathit{then}\ Q \models A \\
P \models \Diamond A & \triangleq & \exists Q.P \xrightarrow{\tau^*} Q\ \mathit{and}\ Q \models A \\
P \models @n & \triangleq & n \in fn(P) \\
P \models \mathbb{K}\phi & \triangleq & P \vdash_k \psi\ \mathit{and}\ \psi \models \phi \\
P \models \mathsf{S}x.A & \triangleq & \exists Q,t.P \equiv (\nu k)Q\ \mathit{and}\ Q \models A\{x \leftarrow t\} \\
& & \mathit{and}\ Q \vdash_k \phi\ \mathit{such\ that}\ t \in \phi \\
& & \mathit{and}\ k \in \mathit{names}(t)
\end{array}
$$

Figure 8: Logic Syntax and Semantics

$Q \mid P$), we can use the free-name predicate to “tag” specific subsystems and reason about their knowledge explicitly: $\Box\mathsf{H}\,\mathit{key}.(@\mathit{tag} \wedge \mathbb{K}\,\mathit{key} \mid \mathbf{T})$ which denotes “it is always the case that an agent with the free name *tag* knows the key” (this subsumes the need for an indexed knowledge operator such as that in [15]).

Notice how the expressiveness of the logic arises from the ability to combine the three types of modalities: dynamic ($\Box, \Diamond$), spatial (H, $\mid$) and epistemic ($\mathbb{K}$). The dynamic connectives allow us to range over a specific execution or all possible executions, the spatial connectives allow us to mention restricted names (usually used to model keys and nonces) and to refer to subsystems, and the epistemic connectives allow us to analyze derivable terms of a process.

The semantics for $\mathbb{K}\phi$ pose a challenge in the sense that they use the notion of knowledge derivation from Section 2.1. While this definition is adequate from a semantic perspective, it makes use of the DY equational closure of a set which is not stable by reduction of terms, and thus doesn’t provide a clear way of algorithmically determining if $\psi \models \phi$. We approach the problem with a purely logical approach and characterize knowledge derivation with a structural proof system for knowledge formulas, unlike the approach of [1].

3.2 Proof System for Knowledge Formulas

Our proof system, formulated as a sequent calculus, is equipped with rules from the equational theory in order to consider the ability to combine terms to generate new information. Each rule of our calculus represents a possible computational step that an agent can perform on terms to produce a new term. Intuitively, if a sequent $\Gamma \vdash \phi$ is derivable, the knowledge formula ϕ is deducible from the knowledge represented by Γ.

Definition 3.1 (Proof System K for Knowledge Formulas) *The sequent calculus formulation of our proof system K for knowledge formulas is defined by the rules of Fig. 9.*

The rules for identity and conjunction are standard. Rule *funRight* states that we are justified in concluding a complex term if we can derive its subterms. Rule *AttLeft* states that all that can be derived from a complex term $f(t_1,\ldots,t_n)$ can also be derived from its subterms; rule *DestrLeft* reflects the equalities of the equational theory: what can be deduced from s can also be deduced from terms equal to s under the equational theory.

$$\frac{}{\Gamma, A \vdash A}\text{(Id)} \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}\ (\wedge\text{: left}) \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (\wedge\text{: right})$$

For every constructor function symbol f with arity n, such that $f \in \Sigma$:

$$\frac{\Gamma \vdash t_1 \ldots \Gamma \vdash t_n}{\Gamma \vdash f(t_1, \ldots, t_n)}\ \text{(funRight)} \qquad \frac{\Gamma, f(t_1, \ldots, t_n) \vdash C}{\Gamma, t_1, \ldots, t_n \vdash C}\ \text{(AttLeft)}$$

For every equation $f(t_1, \ldots, t_n) = s \in E$:

$$\frac{\Gamma, s \vdash C}{\Gamma, f(t_1, \ldots, t_n) \vdash C}\ \text{(DestrLeft)}$$

Figure 9: Proof System for Knowledge Formulas

For the sequent calculus K, we establish the results of *soundness*, *completeness* and *decidability*.

Theorem 3.2 (Soundness of K) *Given a set of terms S and a term A, if $S \vdash A$ then $S \models A$.*
Proof: By induction on the derivation of $S \vdash A$ ■

Theorem 3.3 (Completeness of K) *Given a set of terms S and a term A, if $S \models A$ then $S \vdash A$.*

Theorem 3.4 (Decidability of K) *For any set of terms S and term A, $S \vdash A$ is decidable.*

The proofs of completeness and decidability rely on a finite approximation result for the DY equational closure of a set of terms. More concretely, for each finite set of terms S and equational theory, it is possible to build a finite set $b(S)$ from which all terms in the DY equational closure of S may be determined.

Proposition 3.5 (Approximation of $\mathfrak{F}(S)$) *Let S be a finite set of terms. We may construct in polynomial time an approximation to $\mathfrak{F}(S)$, named $b(S)$, a finite set with the following property:*

$$\forall M \in \mathfrak{F}(S), \exists\, C, \overline{t} \in b(S) \textit{ such that } M = C[\overline{t}]$$

where $C[-]$ is a functional context solely built out of constructors.
Proof: The finite approximation $b(S)$ is built from the terms of S by interpreting the rewrite rules of the theory as contexts of a bounded size. Therefore, applying function symbols to terms of S up to the bound of the context produces a new term by then applying the rewrite rule. This procedure is iterated, eventually reaching a fix-point, due to the subterm convergency property of the equational theories (the idea is that each time we produce a new term, the term will be smaller then the terms used to generate it). The resulting computable set has the property that defines our approximation [17]. ■

The approximation $b(S)$ is such that all terms of $\mathfrak{F}(S)$ can be built from terms of $b(S)$ just by applying constructors, no longer requiring the equations from the theory. Completeness follows from the fact that our proof system is able to emulate the computation steps required to generate the approximation. Given a set of terms S, $b(S)$ is generated by applying functions to terms of S, applying a rewrite rule to the resulting term and iterating. Thus, our proof system is complete since the computation steps of $b(S)$ may be emulated by the rules of proof system K, and we may then apply function symbols to terms of $b(S)$ to produce terms of $\mathfrak{F}(S)$. The latter is trivial due to *funRight* and *AttLeft*. The former we prove through the following lemma.

Lemma 3.6 ***(Completeness of K i.r.t the Approximation).*** *Given a set of terms S ,if $t \in b(S)$ then $S \vdash t$.*
Proof: Through instances of AttLeft *it is possible to apply functions to terms of S up to the bound of the context used in $b(S)$. Through an instance of* DestrLeft *the corresponding rewrite rule can applied, and through* Id *the new term is derived at the root of the proof tree [17].* ■

To emulate the iteration with the proof system, that is, to perform similar computations with S and the new term, the auxiliary result of reasoning with cuts is required.

Lemma 3.7 ***(Cut Admissibility in K).*** *If* $\Gamma \vdash A$ *and* $\Gamma, A \vdash C$ *then* $\Gamma \vdash C$.
Proof: See [17].

Using Cut, the proof system is able to emulate the iterative procedure by building the previously described proof tree that allows the derivation of a new term, and using the new term as the cut formula. This technique can then be applied to produce any term of $b(S)$, as required. Since the computation of $b(S)$ always terminates, Theorem 3.4 holds.

3.3 Model-Checking

We know that model-checking is decidable for the logic without the new modalities [7], for the class of bounded processes. Therefore, we need only show that our two modalities preserve decidability.

Proposition 3.8 (Decidability of model-checking $\mathbb{K}$) *Let* ϕ *be a finite set of terms. Checking that* $P \models \mathbb{K}\phi$ *is decidable.*

The above proposition holds since for any process P it is possible to collect its set of relevant terms ψ ($P \vdash_k \psi$), compute the finite approximation $b(\psi)$ and check that each term in ϕ can be constructed from terms of $b(\psi)$ by application of constructors.

Proposition 3.9 (Decidability of model-checking $\mathsf{S}x.A$) *Checking that* $P \models \mathsf{S}x.A$ *is decidable.*

Decidability of $\mathsf{S}x.A$ follows from the fact that if $P \equiv (\nu n)Q$, it is possible to collect the set ψ of relevant terms of process Q, pick some term t from ψ that contains the name n and check that $Q \models A\{x \leftarrow t\}$. Given that model-checking the core logic with $\mathbb{K}$ is decidable, it follows that checking $P \models \mathsf{S}x.A$ is decidable and therefore model-checking for our logic is decidable for the class of bounded processes.

Theorem 3.10 (Decidability of Model-Checking) *Checking that* $P \models A$ *is decidable for the class of bounded processes.*

4 Expressiveness and Extensions

Having presented our framework, we discuss some extensions to our work that can be used to model and analyze systems. In particular, we discuss the representation of attackers and modeling and verification of correspondence assertions [18] in our framework.

4.1 Modeling Attackers

To analyze a security protocol one usually needs to consider how it behaves in any possible environment. While our logic focuses on the analysis of closed systems, it is possible to verify properties of a system in an arbitrary environment, by internalizing an arbitrary attacker in the system. The general idea is that, for any process P, we may determine a process Q (making essential use of the attacker prefix construct) such that $P|Q$ reaches some state whenever P reaches an equivalent state when placed in an arbitrary environment. While the explicit specification of an attacker for a given protocol may not be easy, our approach to represent the attacking environment is quite different and general, and may indeed be used to find attacks (see example in Section 4.2). We can generically model a Dolev-Yao [12] attacker in our framework by considering the number of message exchanges and the communication channels used in a protocol.

$$\begin{array}{lcl} A(K) & \triangleq & (\nu K_{ab}, N)\, c\langle \mathtt{enc}(\mathtt{pair}(K_{ab}, N), K)\rangle.c(x).[N-1 = \mathtt{dec}(x, K_{ab})] \\ B(K) & \triangleq & c(x).\mathtt{let}\, K_{ab} = \mathtt{fst}(\mathtt{dec}(x,K))\; N = \mathtt{snd}(\mathtt{dec}(x,K))\; \mathtt{in}\; c\langle \mathtt{enc}(N-1, K_{ab})\rangle \\ Sys & \triangleq & (\nu K)\, (A(K) \mid B(K)) \end{array}$$

Figure 10: Modeling the Example

$$\begin{array}{lcl} Attacker & \triangleq & c(x).c\langle * \rangle.c(y).c\langle * \rangle.mem\langle x, y\rangle \\ World & \triangleq & (Sys \mid Attacker) \\ World & \models & \neg \Diamond \mathsf{H}k.(\mathbf{2} \mid (@mem \wedge \mathbb{K}k)) \end{array}$$

Figure 11: An Attacker for the Example

Considering an arbitrary protocol modeled as a process, the role of an attacker is to intercept all communications of the principals and be able to inject any message it can produce, given its knowledge at the time, at any point where a principal expects to receive a message (cf. our attacker output). Thus, a Dolev-Yao attacker consists of a process that for all outputs of the protocol performs an input (storing the received message) and for all inputs performs an attacker output. For instance, consider the following protocol, where K is a shared key, N a fresh value and K_{ab} a session key generated by A:

$$\begin{array}{c} A \rightarrow B : \{K_{ab}, N\}_K \\ B \rightarrow A : \{N-1\}_{K_{ab}} \end{array}$$

In our process model, such a protocol would be represented as done in Fig. 10 (we omit the signature and equational theory). An attacker for this protocol, following our *attacker schema* is presented in Fig. 11. We can then state that it is never the case that the attacker can know one of the keys used in the protocol. While some minor effort of representing an attacker is necessary, we can easily represent a generic attacker for a protocol by following a pre-determined schema.

We currently only consider finite protocols, modeled as processes in our calculus that use a communication channel c as their communication medium (written P_c). We have not pursued infinite protocols as of yet, but we believe it to possible to extend our approach to infinite protocols by defining the attacker as a recursive process with a parallel store (that is used to store the messages of the protocol). To analyze such a system, we would then employ recursive formulas by using the fixpoint operators of the logic.

Our attacker for finite protocols is defined as follows: For each output on c, the attacker performs an input on c (and stores the message). For each input on c, the attacker performs an attacker output.

Definition 4.1 (Attacker Generation Procedure) *Given a process P_c that models a finite protocol, the set S that tracks the attacker memory, an attacker for P can be generated by procedure* $\mathsf{Attacker}(P, S)$ *defined in Fig. 12 (x and m are fresh in P and the attacker).*

The generation procedure produces the necessary actions by inspection of the process dynamics: if an output can occur in the process, the attacker intercepts the message and memorizes it; if an input can occur in the process, the attacker injects any message it can produce from its knowledge; in the case where the protocol has no more actions, we represent the attacker memory with an output $m\langle x_1, \ldots, x_n\rangle$, modeling the attacker's memory throughout the protocol run. We thus show how an attacker can be extracted by inspection of the considered protocol. We can show that this attacker is general in our framework, in the sense that it can simulate the behavior expected from an adversarial medium (c.f. Dolev-Yao attacker). Note that this result does not yet fully apply to our tool implementation, as we discuss later in this section.

```
proc Attacker($P, S$) $\equiv$
  if $P \xrightarrow{\alpha} Q \wedge \alpha =$ input on $c$ then $c\langle * \rangle$.Attacker($Q, S$) fi
  if $P \xrightarrow{\alpha} Q \wedge \alpha =$ output on $c$ then $c(x)$.Attacker($Q, S \cup x$) fi
  if $P \overset{\alpha}{\nrightarrow}$ then $m\langle x_1, \ldots, x_n \rangle$ where $x_i \in S$ fi.
```

Figure 12: Attacker Generation

Definition 4.2 (K-Set) *Given a process P, we define its K-Set K_P, the set of all terms known by P as:* $K_P \triangleq \{t \mid P \models \mathbb{K}\, t\}$

Theorem 4.3 (Monotonicity of K-Sets under Synchronization) *Let P_c and A be a processes such that $P_c \xrightarrow{c\langle M \rangle} P'_c$ and $A \xrightarrow{c(x)} A'$. We have that $K_{A'} \subseteq \mathfrak{F}(K_A \cup M)$.*

We begin with the K-Set of a process, the set of all terms known by the process that we observe in our logic, and we show that the evolution of arbitrary processes' K-Sets through synchronization is monotonic: the resulting process' knowledge will be a subset of the initial process' knowledge, plus any received messages. We state a similar property of our generated attacker's K-Set. Over time, the attacker's K-Set captures all messages exchanged in the protocol.

Theorem 4.4 (Monotonicity of Attacker Storage) *Let P_c and At be processes such that* $At =$ Attacker$(P_c, \{\}, c)$, $P \xrightarrow{c\langle M \rangle} P'$ *and* $At \xrightarrow{c(x)} At'$. *We have that* $K_{At'} = \mathfrak{F}(K_{At} \cup M)$.

Our Attacker Simulation (Lemma 4.5) and Process Knowledge (Lemma 4.6) lemmas provide some insight on the expressiveness of our attacker. Lemma 4.5 shows that a generated attacker, can obtain as much knowledge as an arbitrary process interacting with a finite protocol. Lemma 4.6 states a similar property, regarding the knowledge a finite protocol may obtain while interacting with our attacker.

Lemma 4.5 (Attacker Simulation) *Let P_c and A be processes.*
If $(\nu\overline{n})(P_c \mid A) \longrightarrow (\nu\overline{n})(P'_c \mid A')$ *and* $At =$ Attacker(P_c, S) *with* $K_A \subseteq K_{At}$ *then* $\exists At', S'$ *such that* $(\nu\overline{n})(P_c \mid At) \xrightarrow{*} (\nu\overline{n})(P'_c \mid At')$ *and* $At' =$ Attacker$(P'_c, S \cup S')$ *and* $K_{A'} \subseteq K_{At'}$.

Lemma 4.6 (Process Knowledge) *Let P_c, A be processes and ϕ a knowledge formula.*
If $(\nu\overline{n})(P_c \mid A) \xrightarrow{*} (\nu\overline{n})(P'_c \mid A')$ *and* $P'_c \models \mathbb{K}\phi$ *and* $At =$ Attacker(P_c, S) *with* $K_A \subseteq K_{At}$ *then* $\exists At', S'$ *such that* $(\nu\overline{n})(P_c \mid At) \xrightarrow{*} (\nu\overline{n})(P'_c \mid At')$ *and* $P'_c \models \mathbb{K}\phi$ *and* $At' =$ Attacker$(P'_c, S \cup S')$.

Furthermore, from Lemma 4.6 follows that, in our logic, a finite protocol interacting with an arbitrary process is indistinguishable from one interacting with our attacker. Combining these results, we can show that our attacker can behave as one would expect of an adversarial Dolev-Yao agent.

Theorem 4.7 (Preservation of Satisfaction) *Let P_c and A be processes and A any formula. If* $(\nu\overline{n})(P_c \mid A) \xrightarrow{*} (\nu\overline{n})(P'_c \mid A')$ *and* $P'_c \models A$ *and* $At =$ Attacker(P_c, S) *with* $K_A \subseteq K_{At}$ *then* $\exists At', S'$ *such that* $(\nu\overline{n})(P_c \mid At) \xrightarrow{*} (\nu\overline{n})(P'_c \mid At')$ *and* $P'_c \models A$ *and* $At' =$ Attacker$(P'_c, S \cup S')$.

Notice that this result follows from the fact that message size for the attacker output prefix is unbounded. Our implementation currently bounds the generated message, to ensure tractability, and thus sacrificing completeness. However, as shown in [16], it is possible to compute a finite bound on the message size required to find an attack. The implementation of this result we leave for future work. It is anyway important to note that our method is already *sound and complete* for passive attackers, even for the case of non finite processes (eg. we may consider any finite control system, or bounded in the sense of [7]).

```
parameter attacker_depth = 2;

defproc A(k) = new N in c!(enc(N,k)).c?(x).[dec(x,k)=h(N)].end!(h(N));
defproc B(k) = c?(x).(begin!(dec(x,k)) | c!(enc(h(dec(x,k)),k)));
defproc Sys = new k in (A(k) | B(k));
defproc Attacker = c?(v).c!(*).s!(v);
defproc World = (Sys | Attacker);

defprop begin = <begin!> true;
defprop end = <end!> true;
defprop corrsp = always (end => begin);
check World |= corrsp;
Processing...
* Process World satisfies the formula corrsp *
```

Figure 13: Checking Correspondence in a Toy Protocol

```
...
defproc Sys = new k in (c!(k).(A(k) | B(k)));
defproc Attacker = c?(u).c?(v).c!(*).s!(v,u);
defproc World = (Sys | Attacker);
...
check World |= corrsp;
Processing...
* Process World does not satisfy the formula corrsp *
```

Figure 14: Checking Correspondence in a Broken Toy Protocol

4.2 Modeling Correspondence Assertions

Correspondence assertions are a technique for verifying authentication properties in protocols [18]. The idea is that the model of each principal in a protocol is refined with begin/end events, named *correspondence assertions*, at each stage of an authentication procedure. Authentication will be established if, for every run of the protocol, all end events for each stage are preceded by a matching begin event. To illustrate the idea, consider the following protocol:

$$\begin{array}{ll} A \to B : \{N\}_k; & B \text{ asserts the reception of } N \\ B \to A : \{h(N)\}_k; & A \text{ asserts the reception of } h(N) \end{array}$$

Principals A and B share a symmetric key k, N is a fresh value and h is a one-way hash function. When B receives $\{N\}_k$ it asserts the beginning of the run of the protocol. B sends message $\{h(N)\}_k$ so that A can verify the freshness of the run, by comparing the received value with its own hash of N. If the test succeeds, A asserts the reception of $h(N)$ and the end of the run. To check correspondence, one has to check that every run of the protocol, in the presence of an adversary, would be such that A's end assertion is always preceded by B's begin assertion, that is, A only ends if B was involved in the protocol.

Using our framework, we can model correspondence assertions by representing the assertion as an output on a channel that is irrelevant to the protocol, and then observing the existence of such outputs with our logic. For instance, our example could be modeled as done in Fig. 13 (note the `attacker_depth` parameter set to 2 due to the size of the second message). We can also successfully handle the case where we consider a faulty system that leaks k to the attacker (and thus correspondence does not hold), as presented in Fig. 14.

5 Concluding Remarks and Related Work

In this paper we have introduced a dynamic spatial epistemic logic for a variant of the applied π-calculus aimed at reasoning about security protocols. We explore the application of spatial and epistemic reasoning to the several agents involved in a security protocol, be they principals or adversaries. In our work, we can reason about the knowledge of the several agents of a protocol and how it can evolve over time. Model-checking for the logic is shown to be decidable for an interesting class of processes and cryptographic primitives.

Our framework allows an interesting degree of freedom in the analyses it can perform, not only allowing one to reason directly about knowledge of principals and attackers but also enabling reasoning with correspondence assertions, which is an important addition to the range of available techniques. Moreover, our internalization of attackers, which does not require a complete behavioral specification, is able to accurately emulate the behavior of a Dolev-Yao attacker, enabling reasoning about the dynamics and knowledge of such an attacker.

Finally, the decidability result for our logic allowed us to implement a model-checking algorithm as a proof of concept extension to the SLMC tool. The main difference between the tool and the theory is that our attacker outputs are parametrized with a maximum message size, to bound the state space. This is the main limitation of the current version of our tool, since it does not yet fully capture the expressiveness of our attacker modeling, given that our results employ a more powerful version of the attacker output.

Overall, we have produced an interesting framework for protocol analysis, the first employing dynamic spatial logics. enabling a very natural (yet precise) way of reasoning about security protocols, all the while allowing reasoning with previously established techniques. Note, however, that our tool is merely a proof of concept of the developed framework, not aimed to compete with more mature tools for protocol analysis such as Avispa [4], Scyther [9], Casper [13] or ProVerif [5]. The main point of divergence of our approach and the ones mentioned before is that instead of mainly focusing on a set of built-in properties, we focus on a generic property language (our logic) and explore its expressiveness.

In terms of related logics, Kremer et al. [8] have proposed an epistemic logic for the applied π-calculus. However, their logic lacks the ability to reason about spatial properties, which is a key element in allowing reasoning about individual agents. Their epistemic modalities focus solely on attacker knowledge, not allowing one to state a property such as that of our introductory example where we care about the knowledge of the attacker but also of the agents within the system.

Another closely related logic is Datta et al.'s PCL [10]. PCL is a well established protocol analysis logic that allows one to verify properties of protocols modelled in a CCS style calculus by reasoning about events that occur in traces of the protocol run. While we focus on the combined reasoning about knowledge and spatial distribution of a protocol, PCL is designed to reason about the composition of several protocols and thus its analyses are more sophisticated than ours (reasoning about invariants in the protocol composition interleavings).

Mardare and Priami have also proposed a dynamic epistemic spatial logic [15] without the issues of security in mind. Their logic is hence substantially different from ours, interpreting knowledge as the possibility of observing actions of other processes and not as the ability to know some piece of information. Being based on CCS, such an approach is not suitable for reasoning about the flow of messages within a system, which is one of our main goals.

For future work we wish to further study the problem of attacker representations, aiming at an expressiveness result along the lines of Theorem 4.6 that does not require the attacker to be able to produce a message of an arbitrary size (this should follow from the result of [16]). This result will be key in removing the previously discussed limitation of our tool.

Acknowledgments The first author acknowledges support for this research provided by the Fundação para a Ciência e a Tecnologia through the Carnegie Mellon Portugal Program under Grant SFRH / BD / 33763 / 2009. We thank Mário Pires and Pedro Adão for their interesting comments on some version of this work. We also acknowledge the anonymous reviewers for their comments and suggestions.

References

[1] Martín Abadi & Véronique Cortier (2006): *Deciding knowledge in security protocols under equational theories*. *Theor. Comput. Sci.* 367(1), pp. 2–32.

[2] Martín Abadi & Cédric Fournet (2001): *Mobile values, new names, and secure communication*. In: *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM, New York, NY, USA, pp. 104–115.

[3] Martín Abadi & Andrew D. Gordon (1997): *A calculus for cryptographic protocols: the spi calculus*. In: *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security*, ACM, New York, NY, USA, pp. 36–47.

[4] A. Armando, David A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò & L. Vigneron (2005): *The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications*. In: *CAV*, pp. 281–285. Available at `http://dx.doi.org/10.1007/11513988_27`.

[5] Bruno Blanchet (2001): *An Efficient Cryptographic Protocol Verifier Based on Prolog Rules*. In: *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, IEEE Computer Society, Cape Breton, Nova Scotia, Canada, pp. 82–96.

[6] Michael Burrows, Martin Abadi & Roger Needham (1990): *A logic of authentication*. *ACM Trans. Comput. Syst.* 8(1), pp. 18–36.

[7] Luis Caires (2004): *Behavioral and Spatial Observations in a Logic for the Pi-Calculus*. *FoSSaCS 2004* .

[8] Rohit Chadha, Stéphanie Delaune & Steve Kremer (2009): *Epistemic Logic for the Applied Pi Calculus*. In: D. Lee, A. Lopes & A. Poetzsch-Heffter, editors: *Proceedings of IFIP FMOODS/FORTE'09*, Lecture Notes in Computer Science, Springer, pp. 182–197.

[9] C.J.F. Cremers (2008): *The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols*. In: *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc.*, *Lecture Notes in Computer Science* 5123/2008, Springer, pp. 414–418.

[10] Anupam Datta, Ante Derek, John C. Mitchell & Arnab Roy (2007): *Protocol Composition Logic (PCL)*. *Theor. Comput. Sci.* 172(1-3), pp. 311–358.

[11] Dorothy E. Denning & Giovanni Maria Sacco (1981): *Timestamps in key distribution protocols*. *Commun. ACM* 24(8), pp. 533–536.

[12] Danny Dolev & Andrew C. Yao (1981): *On the security of public key protocols*. Technical Report, Stanford University, Stanford, CA, USA.

[13] Gavin Lowe (1998): *Casper: A Compiler for the Analysis of Security Protocols*. In: *Journal of Computer Security*, Society Press, pp. 53–84.

[14] Étienne Lozes & Jules Villard (2008): *A Spatial Equational Logic for the Applied* Π*-Calculus*. In: *CONCUR '08: Proceedings of the 19th international conference on Concurrency Theory*, Springer-Verlag, Berlin, Heidelberg, pp. 387–401.

[15] Radu Mardare & Corrado Priami (2006): *Dynamic Epistemic Spatial Logic*. Technical Report, Center for Computational and Systems Biology, University of Trento.

[16] Michaël Rusinowitch & Mathieu Turuani (2003): *Protocol insecurity with a finite number of sessions and composed keys is NP-complete*. *Theor. Comput. Sci.* 299(1-3), pp. 451–475.

[17] Bernardo Toninho & Luis Caires (2009): *A Spatial-Epistemic Logic and Tool for Reasoning about Security Protocols*. Technical Report, Departamento de Informática, FCT/UNL.

[18] Thomas Y. C. Woo & Simon S. Lam (1993): *A Semantic Model for Authentication Protocols*. In: *SP '93: Proceedings of the 1993 IEEE Symposium on Security and Privacy*, IEEE Computer Society, Washington, DC, USA, p. 178.