Lazy Mobile Intruders* (Extended Version)

Sebastian Mödersheim, Flemming Nielson, and Hanne Riis Nielson

DTU Compute, Denmark

Tech-Report IMM-TR-2012-013
Revised version January 8, 2013

Abstract. We present a new technique for analyzing platforms that execute potentially malicious code, such as web-browsers, mobile phones, or virtualized infrastructures. Rather than analyzing given code, we ask what code an intruder could create to break a security goal of the platform. To avoid searching the infinite space of programs that the intruder could come up with (given some initial knowledge) we adapt the lazy intruder technique from protocol verification: the code is initially just a process variable that is getting instantiated in a demand-driven way during its execution. We also take into account that by communication, the malicious code can learn new information that it can use in subsequent operations, or that we may have several pieces of malicious code that can exchange information if they "meet". To formalize both the platform and the malicious code we use the mobile ambient calculus, since it provides a small, abstract formalism that models the essence of mobile code. We provide a decision procedure for security against arbitrary intruder processes when the honest processes can only perform a bounded number of steps and without path constraints in communication. We show that this problem is NP-complete.

1 Introduction

Mobile Intruder With *mobile intruder* we summarize the problem of executing code from an untrusted source in a trusted environment. The most common example is executing code from untrusted websites in a web browser (e.g., in Javascript). We trust the web browser and surrounding operating system (at least in its initial setup), we have a security policy for executing code (e.g., on access to cookies in web-browsers), and we want to verify that an intruder cannot design any piece of code that would upon execution lead to a violation of our security policy [12]. There are many similar examples where code from an untrusted source is executed by an honest host such as mobile phones or virtual infrastructures.

* The research presented in this paper has been partially supported by MT-LAB, a VKR Centre of Excellence for the Modelling of Information Technology. The authors thank Luca Viganò and the anonymous reviewers for helpful comments.

Related Problems The mobile intruder problem is in a sense the dual of the *mobile agents* problem where "honest" code is executed by an untrusted environment [3]. The mobile intruder problem has also similarities with the proof-carrying-code (PCC) paradigm [17]. In PCC we also want to convince ourselves that a piece of code that comes from an untrusted source will not violate our policy. In contrast to PCC, we do not consider a concrete given piece of code, but verify that our environment securely executes *every* piece of code. Also, of course, we do *not* require code to be equipped with a proof of its security.

The Problem and a Solution The difficulty in verifying a given architecture for running potentially malicious code lies in the fact that there is an infinite number of programs that the intruder can come up with (given some initial knowledge). Even bounding the size of programs (which is hard to justify in general), the number of choices is vast, so that naively searching this space of programs is infeasible.

Our key observation is that this problem is very similar to a problem in protocol verification and that one may use similar verification methods to address it. The similar problem in protocol verification is that the intruder can at any point send arbitrary messages to honest agents. Also here, we have an infinite choice of messages that the intruder can construct from a given knowledge, leading to an infinitely branching transition relation of the system to analyze. While in many cases we can bound the choice to a finite one without restriction [4], the choice is still prohibitively large for a naive exploration.

In order to deal with this problem of large or infinite search spaces caused by the "prolific" intruder, a popular technique in model checking security protocols is a constraint-based approach that we call *the lazy intruder* [13, 15, 18, 7]. In a state where the intruder knows the set of messages K, he can send to any agent any term t that he can craft from this knowledge, written $K \vdash t$. To avoid this naive enumeration of choices, the lazy intruder instead makes a *symbolic* transition where we represent the sent message by a *variable* x and record the constraint $K \vdash x$. During the state exploration, variables may be instantiated and the constraints must then be checked for satisfiability. The search procedure thus determines the sent message x in a demand-driven, lazy way.

A basic idea is now that code can be seen as a special case of a message and that we may use the lazy intruder to lazily generate intruder code for us. There are of course several differences to the problem of intruder-generated message, because code has a dynamic aspect. For instance the code can in a sense "learn" messages when it is communicating with other processes and use the learned messages in subsequent actions. Another aspect is that we want to consider mobility of code, i.e., the code may move to another location and continue execution there. We may thus consider that code is bundled with its local data and move together with it, as it is the case for instance on migration operations in virtual infrastructures. As a result, when two pieces of intruder-generate code are able to communicate with each other, then they can exchange all information they have gathered. An example is that an intruder-generated piece of code is

able to enter a location, gather some secret information there, and return to the intruder's home base with this information.

Contribution The key idea of this paper is to use the lazy intruder for the malicious mobile code problem: in a nutshell, the code initially written by the intruder is just a variable x and we explore in a *demand driven, lazy* way what this code could look like more concretely in order to achieve an attack.

Like in the original lazy intruder technique, we do not limit the choices of the intruder, but verify the security for the infinite set of programs the intruder could conceive. Also, like in the lazy intruder for security protocols, this yields only a semi-decision procedure for insecurity, because there can be an unbounded number of interactions between the intruder and the environment; this is powerful enough to simulate Turing machines. However by bounding the number of steps that honest processes can perform, we obtain a decision procedure. We show that this problem is NP-complete.

For such a result, we need to use a formalism to model the mobile intruder code—or several such pieces of code—and the environment where the code is executed. In this paper we choose the mobile ambient calculus, which is an extension of common process calculi with a notion of mobility of the processes and a concept of boundaries around them, the ambients. The reason for this choice is that we can develop our approach very abstractly and demonstrate how to deal with each fundamental aspect of mobile code without committing to a complex formalization of a concrete environment such as a web-browser running Javascript or the like. In fact, mobile ambients can be regarded as a "minimal" formalism for mobility. Moreover, it has a well-defined semantics which is necessary to formally prove the correctness of our lazy mobile intruder technique. We therefore avoid a lot of technical problems that are immaterial to our ideas, and neither do we tie our approach to one particular application field.

2 The Ground Model

2.1 The Ambient Calculus

We use the ambient calculus as defined by Cardelli and Gordon [9]. There is a basic version and an extension with communication primitives; we present the ambient calculus right away with communication and only mention that our method also works, mutatis mutandis, for the basic ambient calculus. Fig. 1 contains the syntax of the ambient calculus, and Fig. 2 and 3 give the semantics by defining a structural congruence $\equiv$ and reduction relation $\rightarrow$, respectively. In these figures, we have already omitted some primitives that we do not consider in this paper, namely replication, name restriction, and path constraints; we discuss these restrictions in Sec. 2.5.

The ambient calculus is an extension of standard process calculi with the usual constructs 0 for the inactive process, $P \mid Q$ for the parallel composition of processes P and Q, as well as input $(x).P$—binding the variable x in P—and output $\langle M \rangle$. In addition we have a concept of a process running within

$P, Q ::=$	processes	$M ::=$	capabilities
0	inactivity	x	variable
$P \mid Q$	composition	n	name
$M[P]$	ambient	$in\ M$	can enter M
$M.P$	capability action	$out\ M$	can exit M
$(x).P$	input action	$open\ M$	can open M
$\langle M \rangle$	output action		

Fig. 1. Considered fragment of the ambient calculus.

$$P \equiv P \qquad \frac{P \equiv Q}{Q \equiv P} \qquad \frac{P \equiv Q \quad Q \equiv R}{P \equiv R} \qquad \frac{P \equiv Q}{P \mid R \equiv Q \mid R}$$

$$\frac{P \equiv Q}{M[P] \equiv M[Q]} \qquad \frac{P \equiv Q}{M.P \equiv M.Q} \qquad \frac{P \equiv Q}{(x).P \equiv (x).Q} \qquad P \mid Q \equiv Q \mid P$$

$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid 0 \equiv P$$

Fig. 2. Structural congruence relation.

a boundary, or *ambient*, denoted $n[P]$, and this ambient has the name n. For instance one may model by $m[P \mid v_1[R] \mid v_2[Q]]$ a situation where a process P is running on a physical machine m together with virtual machines v_1 and v_2 that host processes R and Q, respectively. The communication rule (4) in Fig. 3 for instance says that processes can communicate when they run in parallel, but not when they are separated by ambient boundaries. Process can move with the operations *in* n and *out* n according to rules (1) and (2); also one process can dissolve the boundary $n[\cdot]$ of another parallel running ambient by the action *open* n according to rule (3). In all positions where names can be used, we may also use arbitrary capabilities M, e.g., one may have strange ambient names like *in in* n, but this is merely because we do not enforce any typing on the communication rules, and we will not consider this in examples.

We require that in all processes where two input actions $(x).P$ and $(y).P$ occur, different variable symbols $x \neq y$ are used. This is not a restriction since we do not have the replication operator and can therefore make all variables disjoint initially by α-renaming.

2.2 Transition Relation

The definition of the reduction relation $\rightarrow$ in Fig. 3 is standard, however there is a subtlety we want to point out that is significant later when we go to a symbolic relation $\Rightarrow$. The point is that, to be completely precise, the symbols n, m, P, Q, R, P', and Q' in these rules are *meta-variables* ranging over names

$$n[in\ m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R] \quad (1)$$

$$m[n[out\ m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R] \quad (2)$$

$$open\ n.P \mid n[Q] \to P \mid Q \quad (3)$$

$$(x).P \mid \langle M \rangle \to P\{x \mapsto M\} \quad (4)$$

$$\frac{P' \equiv P \quad P \to Q \quad Q \equiv Q'}{P' \to Q'} \qquad \frac{P \to Q}{n[P] \to n[Q]} \qquad \frac{P \to Q}{P \mid R \to Q \mid R}$$

Fig. 3. Reduction relation of the ambient calculus

and processes, respectively. When applying a rule, these variables are supposed to be *matched* with the process they are applied to.

To work with the symbolic approach later more easily, let us reformulate this and make explicit the matching by interpreting rules as *rewriting* rules. In this view, the rules (1)–(4) of Fig. 3 define the essential behavior of the *in*, *out*, and *open* operators and communication, while the other rules simply tell us to which *subterms* of a process the rules may be applied. For instance, the process $M.P$ does not admit a reduction, even if the subterm P does. We can capture that by an *evaluation context* defined as follows:

$C[\cdot]$	::=	context
	$\cdot$	empty context
	$C[\cdot] \mid P$	parallel context
	$M[C[\cdot]]$	ambient context

We define that each rule $r = L \to R$ of the first four rules of Fig. 3 (where the processes L and R have free (meta-) variables on the left-hand and right-hand side) induces a transition relation on *closed* processes as follows: $S \to_r S'$ holds iff there is an evaluation context $C[\cdot]$ and a substitution σ for all the variables of r such that $S \equiv C[\sigma(P)]$ and $S' := C[\sigma(R)]$.[1]

2.3 Ground Intruder Theory

We now define how the intruder can construct processes from a given knowledge K, which is simply a set of *ground* capabilities (i.e. without variables). This model is defined in the style of Dolev-Yao models of protocol verification as the least closure of K under the application of some operators. These operators are encryption and the like for protocol verification, and here they are the following constructors of processes and capabilities (written with their arguments for readability):

$$\Sigma_p = \{0\,,\ P \mid Q\,,\ M[P]\,,\ M.P\,,\ \langle M \rangle\,,\ in\ M\,,\ out\ M\,,\ open\ M\}$$

[1] One may additionally allow here that S' can be rewritten modulo $\equiv$ to match the rules of Fig. 3 precisely, but it is not necessary because when applying further transition rules, this is done modulo $\equiv$.

$$\frac{}{K \vdash M}\ M \in K \quad \text{(Axiom)} \qquad \frac{K \vdash P \quad P \equiv Q}{K \vdash Q}\ \text{(Str.Cong.)}$$

$$\frac{K \vdash_{V_1} T_1 \quad \ldots \quad K \vdash_{V_n} T_n}{K \vdash_{\cup_{i=1}^{n} V_i} f(T_1, \ldots, T_n)}\ f \in \Sigma_p \quad \text{(Public Operation)}$$

$$\frac{}{K \vdash_{\{x\}} x}\ x \in \mathcal{V} \quad \text{(Use variables)} \qquad \frac{K \vdash_V P}{K \vdash_{V \setminus \{x\}} (x).P}\ \text{(Input)}$$

Fig. 4. Ground intruder deduction rules.

We here leave out the input $(x).P$ because it is treated by a special rule.

Fig. 4 inductively defines the *ground intruder deduction relation* $K \vdash_V T$ where K is a set of ground capabilities, T ranges over capabilities and processes, and V is a set of variables such that $V = fv(T)$ the *free variables* of T. We require that the knowledge K of the intruder contains at least one name k_0, so the intruder can always say *something*. For $V = \emptyset$ we also write simply $K \vdash T$. Let $\mathcal{V}$ denote the set of all variable symbols. The (Axiom) and (Str.Cong.) express that the derivable terms contain all elements of the knowledge K and are closed under structural congruence. The (Public Operation) rule says that derivability is closed under all the operators from Σ_p; here the free variables of the resulting term are the union of the free variables of the subterms. The rule (Use variables) and (Input) together allow the intruder to generate processes that read an input and then use it.

As an example, given intruder knowledge $K = \{in\ n, m\}$ we can derive for instance $K \vdash m[(x).in\ n.out\ x.\langle open\ m\rangle]$.

We use the common term "ground intruder" and later "ground transition system" from protocol verification, suggesting we work with terms that contain no variables. However, we allow the intruder to create processes like $(x).P$ where P may freely contain x, and only require that the intruder processes at the end of the day are *closed* terms (without free variables). We may thus correctly call it "closed intruder" and "closed transition system" but we prefer to stick to the established terms.

2.4 Security Properties

We are now interested in security questions of the following form: given an honest process and a position within that process where the intruder can insert some *arbitrary* code that he can craft from his knowledge, can he break a security goal of the honest process? This is made precise by the following definition:

Definition 1. *Let us specify security goals via a predicate attack(P) that holds true for a process P when we consider P to be successfully attacked. We then also call P an* attack state. *Let $C[\cdot]$ be an (evaluation) context without free variables that represents the honest processes and the position where the intruder can*

insert code. Let finally K_0 be a set of ground capabilities. Then the question we want to answer is whether there exist processes P_0 and P such that $K_0 \vdash P_0$, $C[P_0] \to^* P$ and attack(P).

We generalize this form of security questions as expected to the case where the intruder can insert several pieces of code $P_0, \ldots, P_k$ in different locations, and they are generated from different knowledges $K_0, \ldots, K_k$, respectively.

There are many ways to define security goals for the ambient calculus, and we have opted here for state-based safety properties rather than observational equivalences. In fact, the most simple goal is that no intruder process may ever learn a secret name s. We can thus describe an attack predicate that holds true for states where a secret s has been leaked to the intruder. To do that, let us label all output actions $\langle M \rangle$ that are part of the intruder generated code with superscript i like $\langle M \rangle^i$. We formalize that an intruder-generated process has learned the secret s in a state S:

$$\mathsf{leak_s}(S) \text{ iff } \langle s \rangle^i \sqsubseteq S.$$

Here $\sqsubseteq$ denotes the subterm relation.

Another goal is that the intruder code cannot reach a given position of the honest platform. This can be reduced to a secrecy goal—at the destination waits a process that writes out a secret. A more complex goal is containment: a sandbox may host an intruder code and give that code some secret s to compute with, but the intruder code should not be able to get s out of the sandbox. This can again be reduced to secrecy (of another value s') if outside the sandbox a special ambient $k_0[\mathit{open}\ s.\langle s' \rangle]$ is waiting. From this ambient an intruder process (who initially knows the name k_0) can obtain secret s' if it was able to learn s and get out of the sandbox.

Example 1. As an example let us consider the firewall example from [9]:

$$\mathit{Firewall} \equiv w[k[\mathit{out}\ w.\mathit{in}\ k'.\mathit{in}\ w] \mid \mathit{open}\ k'.\mathit{open}\ k''.\langle s \rangle]$$

The goal is that the firewall can only be entered by an ambient that knows the three passwords k, k', and k'' (in fact having capability *open* k instead of k is sufficient). Here the ambient $k[\cdot]$ acts as a pilot that can move out of the firewall, fetch a client ambient (that needs to authenticate itself) and move it into the firewall. Suppose we run $\mathit{Firewall} \mid P$ for some process P that the intruder generated from knowledge K and define as an attack a state in which $\mathsf{leak_s}$ holds. If K includes $\mathit{open}\ k, k', k''$, then we have an attack, since the intruder can generate the process $P \equiv k'[\mathit{open}\ k.k''[(x).\langle x \rangle^i]]$ from K. An attack is reached as follows:

$$\begin{aligned}
&\mathit{Firewall} \mid P \\
&\to w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s \rangle] \mid k[\mathit{in}\ k'.\mathit{in}\ w] \mid P \\
&\to w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s \rangle] \mid k'[k[\mathit{in}\ w] \mid \mathit{open}\ k.k''[(x).\langle x \rangle^i]] \\
&\to w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s \rangle] \mid k'[\mathit{in}\ w \mid k''[(x).\langle x \rangle^i]] \\
&\to w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s \rangle \mid k'[k''[(x).\langle x \rangle^i]]] \\
&\to w[\langle s \rangle \mid (x).\langle x \rangle^i] \\
&\to w[\langle s \rangle^i]
\end{aligned}$$

If the knowledge K from which the intruder process is created does not include *open* k (or k), k' and k'', then no attack is possible. Also containment of the secret s in the firewall holds.

2.5 The Considered Fragment

For the automation, we have made some restrictions w.r.t. the original ambient calculus. The replication operator $!P \equiv P \mid !P$ together with the creation of new names allows for simulating arbitrary Turing machines and thus prevents a decision procedure for security. Similar to the lazy intruder in protocol verification, we thus bound the steps that honest processes can perform and do this by simply disallowing the replication operator for honest processes. Without replication, one of the main reasons for the name restriction operator $\nu n.P$ is gone, since we can α-rename all restricted names so that they are unique throughout the processes. Note that the name restriction is also useful for goals of observational equivalence, which are essential for privacy goals [1, 2] but which we do not consider in this paper.

Note that we do not bound the size of processes that the intruder creates: the derivation relation $K \vdash P$ allows him to make arbitrary use of all constructors. It may appear as if the intruder were bounded because $K \vdash P$ does not include the replication operator either, but this is not true: an attack always consists of a finite number of steps (as violation of a safety property) and thus every attack that can be achieved by an intruder process with replication can be achieved by one without replication (just by "unrolling" the replication as much as necessary for the particular attack). The difference between unbounded intruder processes and bounded honest processes thus stems from the fact that we ask questions of the form: "can a concrete honest process (of fixed size) be attacked by *any* dishonest process (of arbitrary size)?"

We do not need to give the intruder the ability to create arbitrary new names. The reason is that we have no inequality checks in the ambient calculus, i.e., no process can check upon receiving a capability n that it is different from all names it knows (e.g. to prevent replays). Thus, whatever attack works when the intruder uses different self-created names works similarly with always using the same intruder name k_0 that we give the intruder initially.

Finally, the extension of the mobile ambient calculus with communication includes so-called *path constraints* of the form $M.M'$ that can be communicated as messages. Note that this is not ordinary concatenation of messages (which the symbolic techniques we use can easily handle) but sequences of instructions and only after the first has been successfully executed, the next one becomes available, and so the paths cannot be decomposed. Since this includes several problems that would complicate our method, we have excluded them.

3 Symbolic Ambients

We now introduce the symbolic, constraint-based approach that is at the core of this paper. To efficiently answer the kind of security questions we formalized

in the previous paragraph, we want to avoid search the space of all processes that an intruder can come up with. To that end, we use the basic idea of the symbolic, constraint-based approach of protocol verification, also known as the lazy intruder [13, 15, 18, 7].

When an agent in a protocol wants to receive a message of the form t—a term that contains variables—we avoid enumerating the set of all messages that the intruder can generate and that are instances of t (because this set is often very large or infinite). Rather we remember the constraint $K \vdash t$ where K is the set of messages that the intruder knows at the point when he sends the instance of t. We then proceed with states that have free variables, namely the variables of t (and of other messages as they sent and received). The allowed values for these variables are governed by the constraints. For a fixed number of agents and sessions, this gives us a symbolic finite-state transition system. An important ingredient of this symbolic approach is checking satisfiability of the $K \vdash t$ constraints. The complexity of the satisfiability problem has been studied for a variety of algebraic theories of the operators involved, e.g. [10, 11]; in the easiest theory, the free algebra, the problem is NP-complete [18]. One can check satisfiability of the constraints on-the-fly and prune the search tree when a state has unsatisfiable constraints. Thus during the search messages get successively instantiated with more concrete messages in a demand-driven, lazy way. Hence the name.

Now we carry over this idea to the ambient calculus and apply it to the processes that were written by the intruder, i.e. lazily creating the intruder-generated processes during the search. Recall that in the previous section we defined security problems as reachability of an attack state from $C[P_0]$ where $C[\cdot]$ is a given honest agent and $K_0 \vdash P_0$ is any intruder process generated from a given initial knowledge K_0. We could thus simply work with a symbolic state $C[x]$ where x is a variable and we have the constraint $K_0 \vdash x$.

There are some inconveniences attached to using variables like this for representing processes. First, with every transition the process changes and we therefore need to introduce new variables and relate them to the old ones. Second, the processes can learn new information by communication with others, so the available knowledge changes. For these two reasons we follow a more convenient option and simply represent an intruder generated process by writing $\boxed{K}$ where K is the knowledge from which it was created. K is a set of capabilities and intuitively $\boxed{K}$ represents *any* process that can be created from K. If a process contains two occurrences of $\boxed{K}$ for the same K, they may represent different processes. K may contain variables because we will also handle the communication between processes with the lazy intruder technique. We thus extend the syntax of processes P, Q of Fig. 1 by $\boxed{K}$, and we consider symbolic security problems as reachability of a symbolic attack state (defined in Section 3.2) from an initial state $C[\boxed{K}]$ where $C[\cdot]$ is an honest environment that the intruder code is running in.

A symbolic process will also be equipped with constraints which have the following syntax:

$\phi, \psi ::=$	constraints
$K \vdash M$	intruder deduction constraint
$x = M$	substitution
$\phi \wedge \psi$	conjunction

Intuitively, $K \vdash M$ means that capability/message M can been generated by the intruder from knowledge K. In fact, will not use in the symbolic constraints $K \vdash P$ for a process P, since we have no construct for sending processes and all processes the intruder generates are thus covered by the $\boxed{K}$ notation.

Semantics We define the semantics for pairs (S, ϕ) of symbolic processes and constraints as a (usually infinite) set of closed processes. An *interpretation* $\mathcal{I}$ is a mapping from all variables to ground capabilities. We extend this to a morphism on capabilities, processes, and sets of processes as expected, where $\mathcal{I}$ substitutes only free occurrences of variables. We define the model relation as follows:

$$\begin{array}{lll} \mathcal{I} \models K \vdash M & \text{iff} & \mathcal{I}(K) \vdash \mathcal{I}(M) \\ \mathcal{I} \models x = M & \text{iff} & \mathcal{I}(x) = \mathcal{I}(M) \\ \mathcal{I} \models \phi \wedge \psi & \text{iff} & \mathcal{I} \models \phi \text{ and } \mathcal{I} \models \psi \end{array}$$

The semantics of (S, ϕ) is the set of possible instantiation of all variables and intruder code pieces $\boxed{K}$ with closed processes:

$$\begin{aligned} [\![P, \phi]\!] &= \{Q \mid \mathcal{I} \models \phi \wedge Q \in \mathit{ext}(\mathcal{I}(P))\} \\ \mathit{ext}(\boxed{K}) &= \{P \mid K \vdash P\} \\ \mathit{ext}(x) &= \{x\} \\ \mathit{ext}(n) &= \{n\} \\ \mathit{ext}(f(T_1, \ldots, T_n)) &= \{f(T_1', \ldots, T_n') \mid T_1' \in \mathit{ext}(T_1) \wedge \ldots \wedge T_n' \in \mathit{ext}(T_n)\} \end{aligned}$$

Here the T_i range over capabilities and processes and f ranges over all constructors of capabilities and processes. Note the case $\mathit{ext}(x)$ can only occur when processing a subterm of a process where x is bound, so no free variables occur in any $S_0 \in [\![P, \phi]\!]$.

Lazy Intruder Constraint Reduction A decision procedure for satisfiability of $K \vdash M$ constraints can be designed straightforwardly in the style of [15, 7], since we just need to handle the constructors for capabilities, namely *in*, *out*, and *open*, and we have no destructors (or algebraic properties). The only subtlety here is that we have in general several intruder processes that may learn new capabilities independent of each other and may be unable to exchange with each other what they learned—a *multi-intruder problem*. That means we cannot rely on the *well-formedness* assumption often used in the lazy intruder for protocol verification. Suppose the knowledge K in a constraint contains a variable x, then

well-formedness says that there exists an constraint $K_0 \vdash M_0$ with $K_0 \subseteq K$ and M_0 contains x, i.e., x is part of a term the intruder generated earlier. Without this assumption, constraint satisfiability is more difficult to check in general [5], however the main problem is the analysis of knowledge K in constraints. This is not an issue because we have no analysis rules for the intruder here. For more details, see the proof of Theorem 1.

3.1 Symbolic Transition Rules

We now define a symbolic transition relation on symbolic processes with constraints of the form $(S, \phi) \Rightarrow (S', \phi \wedge \psi)$. Note that the constraints are *augmented* in every step, i.e., all previous constraints ϕ remain and new constraints ψ may be added.

We first want to lift the standard transition rules on ground processes of Section 2.2 to the symbolic level. The idea is to replace the rule *matching* defined above with rule *unification*. Recall that above we have essentially defined a transition rule $r = L \to R$ to be applicable to state S if $S = C[\sigma(L)]$ for some substitution σ and evaluation context $C[\cdot]$. For the symbolic level we have that S may contain free variables that need to be substituted as well.

Definition 2 (Lifting). *Let (S, ϕ) be a symbolic state and r a rule that does not contain any variables that occur in (S, ϕ) (which is achieved by α-renaming the rule variables). Define the* lifting of r to the symbolic model *by a transition relation $\Rightarrow_r$ on symbolic states as follows: $(S, \phi) \Rightarrow_r (S', \phi \wedge \psi)$ holds iff there is an evaluation context $C[\cdot]$ and a term T such that:*

- *$S \equiv C[T]$;*
- *σ is a most general unifier of T and L modulo $\equiv$, i.e., $\sigma(T) \equiv \sigma(L)$ and for no generalization τ of σ it holds that $\tau(T) \equiv \tau(L)$; and*
- *$S' = \sigma(C[\sigma(R)])$ and $\psi = eq(\sigma)$*

where $eq(\sigma)$ is the formula $x_1 = t_1 \wedge \ldots x_n = t_n$ if $\sigma = [x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$.

Observe σ may now replace also variables that occur in S and thus σ is applied also to $C[\cdot]$. Moreover for a given (S, ϕ) and rule r there can only be finitely many most general unifiers σ as discussed in the proof of Theorem 1.

Example 2. Using the *in* rule, we can now make the following symbolic transition: $(x[P] \mid y[in\ z.Q], \phi) \Rightarrow (z[P \mid y[Q]], \phi \wedge x = z)$

Similarly, also $(x[P] \mid y[in\ z.\boxed{K}], \phi) \Rightarrow (z[P \mid y[\boxed{K}]], \phi \wedge x = z)$ is possible for an intruder generated piece of code $\boxed{K}$.

So far, however, the rules do not allow us to make an *in* transition on the following state: $(x[P] \mid y[\boxed{K}], \phi)$ even if the intruder can generate a process of the form $in\ z.Q$ from knowledge K. We will see below how to add appropriate rules for intruder-generated processes, so that for instance in the above state a variant of the in-rule is applicable.

It is immediate that the described symbolic transitions are sound (i.e., all states that are reachable in the symbolic model represent states that are reachable in the standard ground model). There are however not yet complete: in the condition $S \equiv C[T]$ above we restrict the application of rule r to contexts that exist in S—without instantiating intruder code like $\boxed{K}$ first. Giving a complete set of rules for intruder processes is the subject of the rest of this subsection.

Intruder-written Code We now come to the very core of the approach: lazily instantiating a piece of code $\boxed{K}$ that the intruder generated from knowledge K with a more concrete term in a demand-driven way. This is basically what is missing after the lifting of the ground rules (Def. 2): when an "abstract" piece of intruder-written code $\boxed{K}$ prevents the application of a rule that would be applicable when replacing $\boxed{K}$ with some more concrete process P such that $K \vdash P$. Obviously we would like to identify such situations without enumerating all processes P that can be generated from K.

In the example $x[P] \mid y[\boxed{K}]$ we discussed above, we have the following possibility: if the intruder code marked $\boxed{K}$ were to have the shape $in\ x.Q$, we could apply the in rule and get to the state $x[y[\boxed{K}] \mid P]$, assuming $K \vdash in\ x$. Note the residual code (inside $y[\cdot]$ after the move) is again something generated from knowledge K.

There is a systematic way to obtain all rules that are necessary to achieve completeness, namely by answering the following question: given a symbolic process with constraints (S, ϕ), any ground process $S_0 \in [\![S, \phi]\!]$, and a transition $S_0 \to S_0'$ what rule do we need on the symbolic level to perform an analogous transition? Thus, we want to reach an $(S', \phi \wedge \psi)$ (in zero or more steps) such that $S_0' \in [\![(S', \phi \wedge \psi)]\!]$. Of course, the rule should also be sound (i.e. all $S_0' \in [\![(S', \phi \wedge \psi)]\!]$ are reachable with ground transition rules from some $S_0 \in [\![(S, \phi)]\!]$). Soundness is relatively easy to see, because we need to consider rules only in isolation. We now systematically derive rules for each case of (S, ϕ), S_0, and S_0' that can occur and thereby achieve a sound and complete set of symbolic transition rules.

Recall that by the definition, for a transition from S_0 to S_0' with rule $r = L \to R$, we need to have an evaluation context $C_0[\cdot]$ and a substitution σ of the rule variables such that $S_0 = C[\sigma(L)]$ and $S_1 = C[\sigma(R)]$.

The symbolic transition rules we have defined above already handle the case that the symbolic state S has the form $S = C'[T]$ where $\sigma(C'[\mathrm{cot}]) = C[\cdot]$ and $\sigma(L) \equiv \sigma(T)$ (as shown in the examples previously) where at a corresponding position a similar rule (under renaming) can be applied without instantiating intruder code. This includes the case that a rule variable P of type process is unified with a piece $\boxed{K}$ of intruder code.

Another case that does not require further work is when the rule match in S_0 is for a subterm of intruder-generated code, i.e. that is subsumed by some $\boxed{K}$ in the symbolic term S. Here we use the fact that intruder deduction is closed under evaluation: if $K \vdash P$ and $P \to P'$, then also $K \vdash P'$.

Therefore all remaining cases that we need to handle are where one or more proper subterms of the redex $\sigma(L)$ in S_0 are intruder-written code that are not trivial, i.e. represent a variable in L. We make a case distinction

- by the different transition rules for $\rightarrow$, namely (1)–(4),
- and by how S relates to the matching subterm in S_0.

In-Rule Let us mark three positions in the *in* rule which could be intruder-written code and that are not yet handled:

$$\overbrace{n[\underbrace{in\ m.P}_{p2} \mid Q]}^{p1} \mid \underbrace{m[R]}_{p3} \rightarrow m[n[P \mid Q] \mid R]$$

In fact, this notation contains a simplification: for instance looking at position p_2, we could also have the variant that the intruder code is of the form $in\ m.P \mid P'$. In such a case, the intruder code piece $\boxed{K}$ in the symbolic state would not exactly correspond to a subterm of the matched rule, but only after "splitting" $\boxed{K}$ into $\boxed{K} \mid \boxed{K}$. Such a splitting rule would obviously be sound, but we do not want to include it, and rather perform such splits only in a demand driven way (as the following cases show)—and to keep the notation simple for the positions in the rules. So all positions indicated here are considered under the possibility that the intruder code itself is a parallel composition; note also we are matching/unifying modulo $\equiv$.

In rule with intruder code at position p_1 The first case we consider is when only at p_1 is intruder code, i.e., we have some intruder code running in parallel with an ambient $m[R]$; then the intruder code may be able to enter m if it has the capability $in\ m$. As said before, we could have the case that the intruder code first splits into two parts and only one part enters m while the other part stays outside. This can be helpful if the intruder code does not have the capability $out\ m$. Since the intruder code can always be trivially 0 if there is nothing to do, it is not a restriction to make the split, so we avoid giving two rules. We obtain:

$$\boxed{K} \mid m[R] \Rightarrow \boxed{K} \mid m[x[\boxed{K}] \mid R] \text{ and } \psi = K \vdash in\ m \wedge K \vdash x \qquad (5)$$

Here we denote with ψ the new constraints that should be added to the symbolic successor state. x is a new variable symbol (that does not occur so far). The reason for introducing this new symbol x is that a process cannot move without being surrounded by an ambient $n[\cdot]$ construct; as the $n[\cdot]$ of the normal *in* rule has now become part of the $\boxed{K}$ code, we need to say that the intruder himself created the ambient. As there is no obligation to pick a particular name for that ambient, we simply leave it open and just require the intruder can construct it from knowledge K. Note that it would be unsound in general to simplify the right-hand side to $m[\boxed{K} \mid R]$ because the intruder cannot get rid of the surrounding $x[\cdot]$ (even though self-chosen) without another process performing $open\ x$.

To see the soundness of this rule, consider that the intruder code matched on the left-hand side of the rule should have the form $P_1 \mid x[in\ m.P_2]$ for some processes P_1 and P_2 generated from knowledge K. These are then represented by the two $\boxed{K}$ pieces on the right-hand side of the rule.

In rule with intruder code at position p_2 Here, intruder code is running inside ambient n that runs in parallel with ambient m. The intruder code can move ambient n into m, if it has the capability $in\ m$:

$$n[\boxed{K} \mid Q] \mid m[R] \Rightarrow m[n[\boxed{K} \mid Q] \mid R] \text{ and } \psi = K \vdash in\ m \tag{6}$$

Note that we could have again the situation that the intruder code is a parallel composition, i.e. of the form $in\ m.P_1 \mid P_2$. However, then after the move we still have $P_1 \mid P_2$ and we thus do not make the split explicit, because this case is still subsumed by $\boxed{K}$ on the right-hand side.

In rule with intruder code at position p_3 Now we consider the situation that an honest ambient $n[in\ m.P \mid Q]$ that wants to enter an ambient m that runs in parallel with intruder code. If the intruder code has name m, it can provide the ambient that the honest process can then enter:

$$n[in\ m.P \mid Q] \mid \boxed{K} \Rightarrow m[n[P \mid Q] \mid \boxed{K}] \mid \boxed{K} \text{ and } \psi = K \vdash m \tag{7}$$

Here we have again an explicit split of the intruder process into two parts. This is because the concrete intruder process that is partially matched by the left-hand side may have the form $m[R_1] \mid R_2$, i.e. not entirely running within m, and we thus need to denote that residual process explicitly on the right-hand side.

In rule with intruder code at several positions If the intruder code is at several positions of the rule, we get the following situations. Obviously we do not need to consider the combination $(p_1) + (p_2)$ because (p_2) is a sub-position of (p_1). The case $(p_1) + (p_3)$ means that we have two intruder processes (in general with different knowledge) to run in parallel: $\boxed{K} \mid \boxed{K'}$. We will show below (when we treat communication) that what they can achieve together is to pool their knowledge and join to one process $\boxed{K \cup K'}$.

What is left is the combination $(p_2) + (p_3)$ which means that one intruder process runs inside an ambient n and that runs in parallel with another intruder process. This case we can express by the following rule:

$$n[\boxed{K} \mid Q] \mid \boxed{K'} \Rightarrow x[n[\boxed{K} \mid Q] \mid \boxed{K'}] \mid \boxed{K'} \text{ and } \psi = K \vdash in\ x \wedge K' \vdash x \tag{8}$$

Note that the two processes that we start with may not have the same knowledge (here K and K'). Again, we have an explicit split on the side of the K'-generated process into a part that is entered by $n[\cdot]$ and one that remains outside. Also, again, this rule has a new variable x for the name of the ambient that is entered by $n[\cdot]$; this name needs to be part of K' while K only needs to have the $in\ x$ capability.

This rule is a problem for the termination of our approach. Observe that the left-hand side ambient $n[\cdot]$ occurs identically as a subterm on the right side; so the rule "packs in" the $n[\cdot]$ ambient into another $x[\cdot]$ ambient. We will therefore later show that we can limit the application of this rule without loosing attacks.

Out Rule For the *out* rule we have two positions of intruder code to consider:

$$m[\overbrace{n[\underbrace{out\ m.P}_{p2} \mid Q]}^{p1} \mid R] \to n[P \mid Q] \mid m[R]$$

Out Rule with intruder code at position p_1 Here we have the situation that the intruder code is within an ambient m and has the capability *out* m. To move parts of the code, the intruder must put it within some ambient x (where x is again a new variable symbol):

$$m[\boxed{K} \mid R] \Rightarrow x[\boxed{K}] \mid m[\boxed{K} \mid R] \text{ and } \psi = K \vdash out\ m \wedge K \vdash x \tag{9}$$

Out rule with intruder code at p_2 This situation is similar except that the intruder code is already contained within an ambient n. We then have:

$$m[n[\boxed{K} \mid Q] \mid R] \Rightarrow n[\boxed{K} \mid Q] \mid m[R] \text{ and } \psi = K \vdash out\ m \tag{10}$$

This subsumes also the case that there is intruder code at both in m and in n (i.e. also within what is matched as R here).

Open-Rule The open rule has also just two positions for intruder code, the opening code and the opened code:

$$\underbrace{open\ n.P}_{p1} \mid \underbrace{n[Q]}_{p2} \to P \mid Q$$

The rules for the intruder code at p_1 and at p_2, respectively are immediate:

$$\boxed{K} \mid n[Q] \Rightarrow \boxed{K} \mid Q \text{ and } \psi = K \vdash open\ n \tag{11}$$

$$open\ n.P \mid \boxed{K} \Rightarrow P \mid \boxed{K} \text{ and } \psi = K \vdash n \tag{12}$$

The case $(p1) + (p2)$ is again the case of two parallel communicating processes that is treated next.

Communication Rule Again there are two possible positions where intruder code could reside, namely as the sender or as the receiver:

$$\underbrace{(x).P}_{p1} \mid \underbrace{\langle M \rangle}_{p2} \to P[x \mapsto M]$$

Communication with the intruder receiving The intruder can receive a message M from an honest process running in parallel:

$$\boxed{K} \mid \langle M \rangle \Rightarrow \boxed{K \cup \{M\}} \tag{13}$$

Here the resulting intruder process has the message M simply added to its knowledge. The idea is that the remaining process can behave like any process that the intruder could have created, if he initially knew $K \cup \{M\}$. To see that this is sound, consider that the intruder process would have the form $(x).P$ for a new variable x that can occur arbitrarily in P. Thus if this process reads M, the resulting $P[x \mapsto M]$ is is a process that can be generated from knowledge $K \cup \{M\}$ if P was created from knowledge K.

Communication with the intruder sending For the case that intruder code sends out a message that is received by an honest process, we can be *truly lazy*:

$$(x).P \mid \boxed{K} \Rightarrow P \mid \boxed{K} \text{ and } \psi = K \vdash x \tag{14}$$

Here, we do not instantiate the message x that is being received, we simply add the constraint that x must be something the intruder can generate from knowledge K. This is in fact the classic case of the lazy intruder—postponing the choice of a concrete message that the intruder sends to an agent. Since the intruder knowledge contains at least one name, there is always "something to say", but what it is will only be determined if the variable x gets unified later upon applying some rule (which can render the $K \vdash x$ constraint unsatisfiable).

Communication with the intruder both sending and receiving Finally we have the rule that was mentioned above already: when two intruder processes meet they can exchange their knowledge and work together further on:

$$\boxed{K} \mid \boxed{K'} \Rightarrow \boxed{K \cup K'} \tag{15}$$

This is sound because every $k \in K \setminus K'$ can be sent from the first to the second process until we have $\boxed{K} \mid \boxed{K \cup K'}$ and then the second part subsumes the first, so we can simplify it to $\boxed{K \cup K'}$. Observe that this rule can also be used when we restrict ourselves to the pure ambient calculus without communication: we then simply have two processes in parallel with capabilities K and K', respectively, and what they can achieve is anything a process with capabilities $K \cup K'$ can achieve (even without communication).

As part of the proof of Theorem 1, we formally show that the set of rules we gave for the symbolic transition system are sound and complete, i.e., they represent exactly the reachable states of the original ground transition system. This proof is found in the extended version of this paper [16], but our systematic development of the rules (i.e., covering each possible case) in this subsection serves as a proof sketch for completeness (and the soundness is straightforward to check for each rule).

3.2 Security Properties in the Symbolic System

Before we can state our main result, we need to formally define the properties we can check for in the symbolic system. Right now, we limit ourselves to secrecy goals as a very basic property, and leave the extension to further security properties for future work.

In general for any property that we want to check, we need to be able to express them for both ground and symbolic states, and these definitions for ground and symbolic states must correspond to each other:

Definition 3. *We say that a predicate* $\mathsf{attack}(S_0)$ *on closed processes* S_0 *and a predicate* $\mathsf{ATTACK}(S, \phi)$ *on symbolic processes* (S, ϕ) correspond *iff for every* (S, ϕ) *it holds that*

$$\mathsf{ATTACK}(S, \phi) \text{ iff exists } S_0 \in [\![S, \phi]\!] \text{ such that } \mathsf{attack}(S_0)$$

Recall that the attack predicate for secrecy on the ground level was defined as $\mathsf{leak}_\mathsf{s}(S_0)$ *iff* $\langle s \rangle^i \sqsubseteq S_0$. *Define the corresponding predicate on the symbolic level:*

$\mathsf{LEAK}_\mathsf{s}(S, \phi)$ *iff exists* K *such that* $\boxed{K} \sqsubseteq S$ *and* $K \vdash s \wedge \phi$ *is satisfiable.*

It is immediate that leak_s and LEAK_s correspond: given any (S, ϕ) then

$$\begin{aligned}
\mathsf{LEAK}_\mathsf{S}(S, \phi) \text{ iff exist } & K, \mathcal{I}.\ \boxed{K} \sqsubseteq S, \quad \mathcal{I} \models K \vdash s \wedge \phi \\
\text{iff exist } & K, \mathcal{I}.\ \boxed{K} \sqsubseteq S, \quad \mathcal{I}(K) \vdash s, \mathcal{I} \models \phi \\
\text{iff exist } & \mathcal{I}, C[\cdot].\ C[\langle s \rangle^i] \in \mathit{ext}(\mathcal{I}(S)) \\
\text{iff exists } & S_0.\ S_0 \in [\![S, \phi]\!] \text{ and } \mathsf{leak}_\mathsf{s}(S_0)
\end{aligned}$$

3.3 Main Result

We can now use the symbolic transition system that we have developed using the lazy intruder technique to give a decision procedure for secrecy in our fragment of the ambient calculus without bounding the intruder.

Theorem 1. *The following problem is NP-complete. Given*

- *a name* s,
- *a closed process* $C_0[\cdot]$ *(in our the supported fragment),*
- *and a finite set* K *of ground capabilities as initial intruder knowledge;*

exist P *and* S_0 *such that* $K \vdash P$, $C_0[P] \to^* S_0$ *and* $\mathsf{leak}_\mathsf{s}(S_0)$*?*

Proof. The proof consists of several parts:

1. We first show that the symbolic transition system is sound and correct in that it represents the same set as the ground one, i.e.

$$\{S_0' \in [\![S', \phi']\!] \mid (S, \phi) \Rightarrow^* (S', \phi')\} = \{S_0' \mid S_0 \in [\![S, \phi]\!], S_0 \to^* S_0'\}\ .$$

   The soundness can be proved for each rule individually, while for completeness we show that for every $S_0 \to S_0'$ we can find a corresponding rule on the symbolic level.

2. We show that satisfiability of the deduction constraints ϕ is NP-complete; from this we further derive that our main problem of attack state reachability is NP-hard.
3. It follows that there is a P and S_0 such that $K \vdash P$, $C_0[P] \rightarrow^* S_0$ and $leak_s(S_0)$ iff there is a symbolic state (S', ϕ') such that $(C_0[\boxed{K}], \mathsf{true}) \Rightarrow^* (S', \phi')$ and $LEAK_s(S', \phi')$.
4. We show how to bound the exploration of symbolic state space such that we find an attack if one is reachable at all and such that the length of traces in the restricted space is bounded by a polynomial. It follows that the attack reachability problem is in NP.

Soundness First, we show the soundness of the lifting of every standard rule $r = L \rightarrow R$ to the symbolic approach of Def. 2. Let $(S, \phi) \Rightarrow_r (S', \phi \wedge \psi)$. Consider any interpretation $\mathcal{I} \models \phi \wedge \psi$. Then $\mathcal{I}(S) \rightarrow_r \mathcal{I}(S')$ if we treat all $\boxed{K}$ terms as normal closed processes. Instantiating these $\boxed{K}$ terms with arbitrary closed processes $P \dashv K$ we obtain two closed processes $S_0 \in [\![S, \phi]\!]$ and $S_0' \in [\![s', \phi \wedge \psi]\!]$ such that $S_0 \rightarrow_r S_0'$. Since we chose an arbitrary interpretation with $\mathcal{I} \models \phi \wedge \psi$ and an arbitrary extension of the $\boxed{K}$, $[\![S', \phi \wedge \psi]\!]$ contains only states that are indeed reachable from some $S_0 \in [\![S, \phi]\!]$.

For what concerns the other rules, we can reduce them to standard cases by instantiating the intruder processes $\boxed{K}$ in an appropriate way. For instance, consider again the rule (5):

$$\boxed{K} \mid m[R] \Rightarrow \boxed{K} \mid m[x[\boxed{K}] \mid R] \text{ and } \phi = K \vdash in\ m \wedge K \vdash x$$

Consider that we instantiate in the left-hand side of the rule the intruder code $\boxed{K}$ with the code of the form $P_1 \mid x[in\ m.P_2]$ for some processes P_1 and P_2. This requires that $K \vdash P_1 \mid x[in\ m.P_2]$, which in turn requires that from K the intruder can derive P_1, P_2, x and $in\ m$. Then we have the process $P_1 \mid x[in\ m.P_2] \mid m[R]$ and can apply the symbolic version of the standard in rule to get to the process $P_1 \mid m[x[P_2] \mid R]$. Since P_1 and P_2 are processes generated from K, we have $\boxed{K} \mid m[x[\boxed{K}] \mid R]$ i.e. the right-hand side of (5) with the additional constraints $K \vdash in\ m$ and $K \vdash x$.

Similarly we have the other rules (displaying only the left-hand side with intruder code instantiated appropriately, and the P_i are intruder-generated processes):

(6) $n[in\ m.P_1 \mid Q] \mid m[R]$
(7) $n[in\ m.P \mid Q] \mid m[P_1] \mid P_2$
(8) $n[in\ x.P_1 \mid Q] \mid x[P_2] \mid P_3$.
(9) $m[x[out\ m.P_1] \mid P_2 \mid R]$
(10) $m[n[out\ m.P_1 \mid Q] \mid R]$
(11) $open\ n.P_1 \mid n[Q]$
(12) $open\ n.P \mid n[P_1] \mid P_2$

(13) $(x).P_1 \mid M$ where now $K \vdash_{\{x\}} P_1$, i.e. P_1 is a process with a free variable x that will get instantiated with the capability M. It follows that $K \cup \{M\} \vdash P_1[x \mapsto M]$, hence the result can be written as $\boxed{K \cup \{M\}}$.

(14) Consider the reduction $(x).P \mid \langle M \rangle \mid P_1 \to P[x \mapsto M] \mid P_1$. In rule (14) we have this situation where $\langle M \rangle \mid P_1 \dashv K$ is the intruder process; thus $K \vdash M$. This can then be represented by the transition

$$(x).P \mid \boxed{K} \to P \mid \boxed{K}$$

where the right-hand side has the constraint $K \vdash x$, because according to the semantics all occurrences of x in P on the right-hand side are then instantiated with some term M that can be generated from K. Note also that we required above that for all read operations, the variable names are disjoint, i.e., in all other processes outside P the variable x does not occur (where it in general may *not* be bound to the same term M).

(15) Simply observe that from $K \vdash P_1$ and $K' \vdash P_2$ follows $K \cup K' \vdash P_1 \mid P_2$.

Completeness We now need to show the following: given a symbolic state (S, ϕ), a represented ground state $S_0 \in [\![S, \phi]\!]$, and a transition $S_0 \to S_0'$, we can reach a symbolic state that covers S_0': $(S, \phi) \Rightarrow^* (S', \phi \wedge \psi)$ with $S_0' \in [\![S', \phi \wedge \psi]\!]$.

To make notation easier for this proof, let us assume that we replace all pieces of intruder code $\boxed{K}$ in S be with distinguished variables like $\mathbb{P}$; also let us extend interpretations to these variables, i.e. such that $\mathcal{I}(\mathbb{P}) = P$ for a closed process $P \dashv K$ if K is the intruder knowledge associated with $\mathbb{P}$. Let us thus fix an interpretation $\mathcal{I}$ with $S_0' = \mathcal{I}(S)$.

We now have that $S_0' \equiv C[\sigma(L)]$ and $S_0' = C[\sigma(R)]$ for some rule $L \to R$, substitution σ, and evaluation context $C[\cdot]$. Before we distinguish the cases of the different rules, let us first consider how the position of the match $\sigma(L)$ relates to S:

- *Outside all intruder code*: a corresponding redex exists in S without instantiating any of the intruder variables $\mathbb{P}$. Formally: $S \equiv C_1[T]$ where $\mathcal{I}(C_1[\cdot]) = C[\cdot]$ and $\mathcal{I}(T) = \sigma(L)$. Therefore the lifted version of $(L \to R)$ to the symbolic system (Def. 2) is applicable to (S, ϕ), covering the transition to S_0'.
- *Within intruder code*: the redex corresponds in S to a position within an intruder process $\mathbb{P}$. Formally $C[\cdot] = C_0[C_1[\cdot]]$ for some contexts $C_0[\cdot]$ and $C_1[\cdot]$, $S \equiv C_2[\mathcal{I}(\mathbb{P})]$) for some variable $\mathbb{P}$ and context $C_2[\cdot]$ with $\mathcal{I}(C_2[\cdot]) = C_0[\cdot]$. Thus, this is an internal deduction of an intruder process $P = \mathcal{I}(\mathbb{P})$ i.e. $S_0' = C_0[P']$ for some $P \to P'$. The idea is that in this case $[\![S, \phi]\!]$ also covers S' because intruder deduction is closed under $\to$, i.e., from $K \vdash P$ and $P \to P'$ follows $K \vdash P'$. To see this, one simply shows that from $K \vdash L$ follows $K \vdash R$ for every ground rule $L \to R$. For instance, for the *in* rule, we have: $K \vdash n[in\ m.P \mid Q] \mid m[R]$ can only hold true iff from K we can derive n, m, P, Q, and R. Then also $K \vdash m[n[P \mid Q] \mid R]$. The only tricky rule is (4): $K \vdash (x).P \mid M$ can only hold if $K \vdash_{\{x\}} P$ and $K \vdash M$. Therefore

replace in the proof of $K \vdash_{\{x\}} P$ all occurrences of x with M to obtain the proof $K \vdash P\{x \mapsto M\}$.
- *Overlapping with intruder code*: in all other cases the redex corresponds in S to a position that is neither entirely intruder code nor entirely honest code. (It may however be of the form $\mathbb{P} \mid \mathbb{P}'$.) The rest of the completeness proof is concerned with these cases.

Let us now focus on the smallest subterm of S that corresponds to the redex, i.e. $S \equiv C_1[T]$ such that $\sigma(L) \sqsubseteq \mathcal{I}(T)$. Note that T itself may not be the redex, e.g. we could have $\mathbb{P} \mid m[R]$ and $\mathcal{I}(\mathbb{P}) = P_0 \mid n[in\ m.P_1]$ so only part of T (but not a proper subterm of T) is the redex. We distinguish by the different rules.

In-Rule We have here the situation that $\mathcal{I}(T)$ has a subterm of the form $n[in\ m.P \mid Q] \mid m[R]$ (modulo $\equiv$) while process variables in T prevent the application of this rule. We thus have one of the following situations in the current state S:[2]

- $T \equiv n[\mathbb{P} \mid Q] \mid m[R]$ and $\mathcal{I}(\mathbb{P}) = in\ m.P_1 \mid P_2$ (where $P_2 = 0$ is possible.) This case is thus covered by (5).
- $T \equiv \mathbb{P} \mid m[R]$ and $\mathcal{I}(\mathbb{P}) = P_0 \mid n[in\ m.P_1 \mid P_2]$. This case is covered by (6).
- $T \equiv n[in\ m.P \mid Q] \mid \mathbb{P}$ where $\mathcal{I}(\mathbb{P}) = m[P_1] \mid P_2$: covered by (7).
- $T \equiv \mathbb{P} \mid \mathbb{P}'$: covered by (15).
- $T \equiv n[\mathbb{P} \mid Q] \mid \mathbb{P}'$ where $\mathcal{I}(\mathbb{P}) = in\ x.P_1$ and $\mathcal{I}(\mathbb{P}') = x[P_2] \mid P_3$: this is covered by the rule (8).

Out Rule Here $\mathcal{I}(T) \sqsupseteq m[n[out\ m.P \mid Q]]$. We thus have one of the following situations in the current state S:

- $T \equiv m[\mathbb{P} \mid R]$ and $\mathcal{I}(\mathbb{P}) = n[out\ m.P_0 \mid P_1]$: (9) is applicable.
- $T \equiv m[n[\mathbb{P} \mid Q] \mid R]$ and $\mathcal{I}(\mathbb{P}) = out\ m.P_0 \mid P_1$: (10) is applicable.

Open-Rule Here $\mathcal{I}(T) \sqsupseteq open\ n.P \mid n[Q]$. We thus have one of the following situations in the current state S:

- $T \equiv \mathbb{P} \mid n[Q]$ and $\mathcal{I}(\mathbb{P}) = P_0 \mid open\ n.P_1$: covered by (11).
- $T \equiv open\ n.P \mid \mathbb{P}$ and $\mathcal{I}(\mathbb{P}) = n[P_1] \mid P_2$: covered by (12).
- $T \equiv \mathbb{P} \mid \mathbb{P}'$ where $\mathcal{I}(\mathbb{P}) = openn.P_1$ and $\mathcal{I}(\mathbb{P}') = n[P_2] \mid P_3$ is covered by (15).

Communication Rule Here $\mathcal{I}(T) \sqsupseteq (x).P \mid \langle M \rangle$. We thus have one of the following situations in the current state S:

- $T \equiv \mathbb{P} \mid \langle M \rangle$ and $\mathcal{I}(\mathbb{P}) = (x).P_1 \mid P_2$ Let K be the knowledge of $\mathbb{P}$, i.e. $\mathcal{I} \models K \vdash \mathbb{P}$. Thus $\mathcal{I}(K) \vdash_{\{} x\}P_1$. Therefore $\mathcal{I}(K \cup \{M\}) \vdash P_1\{x \mapsto \mathcal{I}(M)\}$. Thus $\mathcal{I} \models K \cup \{M\} \vdash P_1\{x \mapsto M\}$. Thus, covered by (13).

[2] Slight simplification: instead of using new variables n', m', P', Q', and R' for the subterms of T that correspond to the rule variables n, m, P, Q, respectively, after the match, we directly use the rule variables.

- $T \equiv (x).P \mid \mathbb{P}$ where $\mathcal{I}(\mathbb{P}) = \langle(\rangle M) \mid P_1$. Since x cannot occur in the rest, and $\mathcal{I} \models K \vdash M$ for the knowledge K of $\mathbb{P}$, let $\mathcal{I}' = \mathcal{I}\{x \mapsto M\}$ and let $(S', \phi \wedge \psi)$ be the state that results from (14); then $\mathcal{I}'(S') = S'_0$ and $\mathcal{I}' \models \phi \wedge \psi$.
- $T \equiv \mathbb{P} \mid \mathbb{P}'$ and $\mathcal{I}(\mathbb{P}) = (x).P_1 \mid P_2$ and $\mathcal{I}(\mathbb{P}') = \langle M \rangle \mid P_3$. Let K be the knowledge for $\mathbb{P}$ and K' the knowledge for $\mathbb{P}'$. Then the resulting closed process $P_1\{x \mapsto M\} \mid P_2 \mid P_3$ can be generated from $\mathcal{I}(K \cup K')$, thus covered by (15).

Constraint Satisfiability is NP-Complete We now turn to the constraints of the form $K \vdash M$ that we use in the symbolic approach. It is well-known that satisfiability of such constraints in protocol verification in an NP-complete problem [18]. For the lazy mobile intruder, the class of constraints we get is incomparable to that of protocol verification, as it is in one regard simpler and in another more general. First, the aspect where it is simpler is that we have here only constructors on the intruder knowledge, namely *in*, *out*, and *open* (and the constructors of processes), but no destructors or analysis rules, i.e. the intruder has no operation to obtain a subterm from a term he knows. Second, in one aspect our problem is more general because the intruder process may split into several processes that learn independently and therefore the standard *well-formedness assumption* does no longer hold. This assumption says that the intruder knowledge monotonically grows and variables that occur on the knowledge side of a constraint must originate on the right-hand side of an earlier constraint (i.e., one with smaller knowledge). Intuitively, if the intruder knowledge contains a variable than it represents a choice that the intruder made earlier. As an example that non-well-formed constraints can arise from lazy mobile intruder, consider the process: $\boxed{K} \mid (x).in\ n.\langle x \rangle \mid n[\boxed{K'}]$, which can reach the state $\boxed{K} \mid n[\boxed{K' \cup \{x\}}]$ and the constraint $K \vdash x$. Here we have a variable x in the knowledge of a process $\boxed{K' \cup \{x\}}$ that does not necessarily originate from a subset of K' (if K is not a subset of K').

Containment in NP There is a more general result, i.e. that satisfiability a larger class of non-well-formed constraints is in NP [5, 14]. We anyway briefly sketch the proof for our class, because it works in a different way that is the basis for a more efficient implementation.

The idea is to give a simple proof calculus for satisfiability of the constraints that the lazy mobile intruder can generate. Note that all terms in these constraints are solely built from constants (names), variables, and the operators *in*, *ou*, and *open*. Also we have variable origination: we can order the constraints such that every variable in a knowledge K of a constraint first occurs in the term t to generate in an earlier constraint.

The rules of our proof calculus have the form $\dfrac{\psi}{\phi}$ where $\psi \models \phi$ holds (soundness of the rules). We use them backwards: to show that ϕ is satisfiable, one possible proof is to show that ψ is satisfiable.

$$\frac{K \vdash t \wedge \phi}{K \vdash f\ t \wedge \phi}\ (Generate)\ f \in \{in, out, open\}$$
$$\frac{\sigma(\phi) \wedge eq(\sigma)}{K \vdash t \wedge \phi}\ (Unify)\ s \in K, t \notin \mathcal{V}, \sigma \in mgu(s,t)$$

where *mgu* is the set of most general unifiers of s and t (which is either singleton, or empty—then the rule is not applicable). Intuitively, the first rule says that for composing $f\ t$, it is sufficient to compose t; and the second rule that whenever the term t to compose is unifiable with a known term s, nothing is left to do in any model that is an instance of the unifier σ. The soundness of the rules (i.e. the assumption implies the conclusion) is straightforward.

Let us call a constraint *simple* if $t \in \mathcal{V}$ for every $T \vdash t$ conjunct. Simple constraints are always satisfiable (e.g., instantiate all variables with the initially known name k_0). Note neither rule is applicable to a simple constraint.

For completeness we thus show: any satisfiable constraint is either simple or admits the application of a rule so that the resulting constraint is still satisfiable. To that end consider a satisfiable constraint ϕ and a model $\mathcal{I} \models \phi$. For every conjunct $T \vdash t$ we can label t with a ground derivation of $\mathcal{I}(t)$ from $\mathcal{I}(T)$. Let now $T \vdash t$ be a conjunct of ϕ where $t = f\ t'$. It is straightforward that depending on the last derivation step for $\mathcal{I}(t)$ we can either apply the unify or generate rule and label the resulting constraints again according to $\mathcal{I}$, i.e., $\mathcal{I}$ is still supported by the resulting constraint.

The length of a derivation is polyniomially bounded: let (k, w) be a measure where k is the number of variables in a constraint and w is the sum of the weight of all terms in the constraint (variables and constants having weight 1, and each operator (*in*, *out*, *open*) increases the weight by 1). Define $(k, w) > (k', w)$ iff $k > k'$ or $(k = k'$ and $w > w')$. Then every application of a step reduces the measure (k, w) (either substituting a variable or reducing terms). The increase of the second component for every reduction of k is polynomial because we have only a unary operator. From the calculus we thus obtain a non-deterministic polyniomal time algorithm for checking satisfiability of the constraints by the lazy mobile intruder.

NP-hardness NP-hardness of the constraint satisfaction problem is shown by reduction of satisfiability of Boolean formulae. Let F be Boolean formula with n variables $x_1, \ldots, x_n$ in conjunctive normal form.

We translate this formula into a lazy mobile intruder constraint as follows. We first introduce the variables $px_1, \ldots, px_n$ and $nx_1, \ldots, nx_n$ and the following constraints for every $1 \leq i \leq n$:

$$\{t, f\} \vdash px_i \wedge \{t, f\} \vdash nx_i \wedge \{px_i, nx_i\} \vdash t \wedge \{px_i, nx_i\} \vdash f$$

Here, the names t and f represent true and false, and we thus ensure that each px_i and nx_i is either instantiated with t or f, and that for each i not both px_i and nx_i can be t. Then it is now straightforward to encode each clause

$C = L_1 \vee \ldots \vee L_k$, where each literal L_j is either x_i or $\neg x_i$ for some variable x_i. Let $\hat{L_j} = px_i$ if $L_j = x_i$ and $\hat{L_j} = nx_i$ if $L_k = \neg x_i$:

$$\{\hat{L_1}, \ldots, \hat{L_k}\} \vdash t$$

This ensures that at least one of the $\hat{l_i}$ is t in the original choice. The conjunction of all the constraints produced this way has thus a solution if the original formula F has. Note that this also holds if every intruder knowledge initially contains the constant k_0.

Reachability NP-hard Note that NP-hardness of the constraint satisfaction problem does not directly prove the main problem is NP-hard. In fact, we can show that even without constraint reduction, just by the non-determinism of ambients, we have NP-hardness. To see that consider again a boolean formula $F = C_1 \wedge \ldots \wedge C_m$ in conjunctive normal form and variables $x_1, \ldots, x_n$. Let us first introduce names $t_1, \ldots, t_n, f_1, \ldots, f_n$ where t_i later shall mean x_i is true, and f_i shall mean that x_i is false. We translate F into the following process:

$$(\ |\ {}^{n}_{i=1} w[\langle t_i \rangle \mid \langle f_i \rangle \mid (x_i).k_0[out\ w.\langle x_i \rangle]]) \mid \boxed{\{k_0\}} \mid \hat{C_1} \mid \ldots \mid \hat{C_m}$$

where $\hat{C_j}$ is the translation of clause $C_j = L_{j,1} \vee \ldots \vee L_{j,l_j}$:

$$C_j = w[\langle \hat{L}_{j,1} \rangle \mid \ldots \mid \langle \hat{L}_{j,l_j} \rangle \mid (y_j).k_{j-1}[out\ w.y_j[\langle k_j \rangle]]]$$

where $\hat{L}_{j,l} = \begin{cases} t_k & \text{if } L_{j,l} = x_k \\ f_k & \text{if } L_{j,l} = \neg x_k \end{cases}$.

Now, the first parallel processes non-determinstically choose each x_i to become either t_i or f_i; after the *out* w, the intruder can thus learn these values since he can open k_0. Next, the $\hat{C_k}$ clauses non-deterministically choose one of the literals in C_k and instantiate y_k with t_i if the chosen literal is positive, or f_i if the chosen literal is negative. Among the possible instantiations of y_k is a value that the intruder knows iff the instantiations of the x_i makes one of the literals true, and thus also the clause C_k. Now the process forms an ambient of name k_{j-1} that moves out of the w ambient; this ambient then has the form $k_{j-1}[y_j[\langle k_j \rangle]]$. Thus, if the intruder knows k_{j-1} and y_j, he can obtain k_j. Initially he knows k_0, therefore he can get successively all the k_i iff the x_i form a satisfying solution (and each $\hat{C_k}$ chooses an appropriate literal for y_j as explained above). Thus there is a reachable state in which the intruder learns the last name k_m iff F is satisfiable. Thus, even for an intruder who does not move and who only passively listens, the problem is NP-hard.

Termination The symbolic transition system we have described is finitely branching, i.e., every state has finitely many successor states. For this, note that unification modulo $\equiv$ is finitary (there is a finite set of most general unifiers in each case). For this, recall that parallel composition is associative, commutative

and has a neutral element, for which unification is known to be finitary [6]. It is straightforward to extend this to a finitary unification algorithm for $\equiv$, since the other constructs like $n[\cdot]$ can be treated as free symbols.

However, the symbolic transition system it can still have infinite depth, i.e. it contains infinite traces of the form $(S_1, \phi_1) \to (S_2, \phi_2) \to \ldots$. In order to obtain a decision procedure, we need to show that we can safely stop at a depth bound D, i.e., if there no attack states within depth D, then there are none at all. Showing that D is polynomial in the size of the problem implies that the entire problem is in NP.

For a given problem instance, let N be the maximum number of steps honest processes can make (e.g. $in\ n.m[(x).open\ x]$ has $N = 3$). Let l be the number of locations $n[\cdot]$ in the initial honest process. We prove that the depth D we need to consider can be bounded by $O(k \cdot N \cdot l)$. In fact, rather than giving a precise depth it is more convenient to define when we can stop searching and show that this is bounded by $O(N \cdot l^2)$.

We first consider all the symbolic rules that create a new ambient $x[\cdot]$, namely (5),(8), and (9). Observe that in any (even infinite) trace at most N times any such $x[\cdot]$ can be entered, exited, or opened by an honest agent performing $in\ M$, $out\ M$, or $open\ M$ and unifying M with x. Note that in several cases the intruder needs to build such a structure for moving his own code. If we thus look at where in a symbolic trace these variables x can get instantiated, it can be at most N times caused by an honest agent; also the intruder can enter, exit, or open its own code. Thus, analogous transitions would also work if all the variables x that are not instantiated by the honest agents were instantiated with a name k_0 that is contained in the initial intruder knowledge. This means, we could basically make the same transitions, but the choice of some intruder names would be k_0. Obviously that makes no difference to the $\mathsf{LEAK_s}$ predicate, but other attack predicates may be affected. Moreover, the $k_0[\cdot]$ ambients only help the intruder for moving in or out of an ambient, i.e. all other occurrences of $k_0[\cdot]$ are redundant and one can find a simpler leak without them.

With this observation we can tame the effects of rule (8): we can limit it to N applications in any trace, because all other occurrences can only be necessary for a transition where the intruder needs to surround code by a new ambient in order to move in or out of somewhere, i.e., what is already covered by the rules (5) and (9).

These rules (5) and (9) are also somewhat problematic because of the potentially unbounded creation of new ambients. Let us therefore look at the case that an intruder ambient enters (and similar, exits) another ambient that already contains intruder code. The first case is $\boxed{K} \mid m[\boxed{K'} \mid R]$. The rule (5) would create the state $S = \boxed{K} \mid m[x[\boxed{K}] \mid \boxed{K'} \mid R]$ with constraints $K \vdash x, in\ m$. Using open and communicate instantiating x with k_0 (that is also in K'), we can get to the state $S' = \boxed{K} \mid m[\boxed{K \cup K'} \mid R]$. Since $K \cup K' \supseteq K$, this still entails the state S and it is thus no restriction to go right to S', i.e. define a new

rule

$$\boxed{K} \mid m[\boxed{K'} \mid R] \Rightarrow \boxed{K} \mid m[\boxed{K \cup K'} \mid R] \text{ and } \phi = K \vdash in\ m$$

and to say that the original (5) cannot be applied if this one can. Note that in this variant, only the intruder knowledge inside the m ambient is increased.

A similar case is we have in the state $\boxed{K} \mid m[y[\boxed{K'}] \mid R]$ with a variable y. In this case our (5) rule gives $S = \boxed{K} \mid m[x[\boxed{K}] \mid y[\boxed{K'}] \mid R]$ and $K \vdash in\ m, x$. Here the procedure is more complicated: with (9) we get to the term $\boxed{K} \mid m[x[\boxed{K}] \mid z[\boxed{K'}] \mid y[\boxed{K'}] \mid R]$, with (6), (11), and (15) we get $\boxed{K} \mid m[x[\boxed{K \cup K'}] \mid y[\boxed{K'}] \mid R]$ and again with the same rules to $S' = \boxed{K} \mid m[y[\boxed{K \cup K'}] \mid R]$. Again S' subsumes the state S. We can thus further have the rule

$$\boxed{K} \mid m[x[\boxed{K'}] \mid R] \Rightarrow \boxed{K} \mid m[x[\boxed{K \cup K'}] \mid R] \text{ and } \phi = K \vdash in\ m$$

that is applied instead of (5) whenever possible, just monotonically increasing the knowledge at $\boxed{K'}$.

Similarly for the out rules we thus have two special rules (with similar justifications) that must be taken instead of (9) whenever possible:

$$m[\boxed{K} \mid R] \mid \boxed{K'} \Rightarrow m[\boxed{K} \mid R] \mid \boxed{K \cup K'} \text{ and } K \vdash out\ m$$
$$m[\boxed{K} \mid R] \mid x[\boxed{K'}] \Rightarrow m[\boxed{K} \mid R] \mid x[\boxed{K \cup K'}] \text{ and } K \vdash out\ m$$

Note that this four new rules can always be applied greedily: it never hurts to increase the intruder knowledge (i.e. the semantics of a symbolic state increases). Moreover the application of these greedy rules is limited, since there are only l locations of honest agents, and at most the same number created by the intruder, and the intruder knowledge at each location can hold at most $o \leq N$ capabilities (that can be communicated by honest processes).

The rules (7), (11), (12), (13), and (14) as well as the lifting of the standard rules (Def. 2) can be applied at most N times in total since they "consume" an action of an honest process. The only rules that can still potentially be applied infinitely many times are rules (6) and (10). However as they are just restructuring the process, not removing or introducing constructs, these two rules can be applied in a sequence only a l times before repetition occurs (i.e. a state that could be reached with at most l transitions).

Since we can bound all transitions except (6) and (10) by $O(N \cdot l)$, and we may have up to l steps in between each of them, we arrive $O(N \cdot l^2)$.

3.4 Examples

Let us reconsider the firewall example from before, and see how a lazy intruder process would find the attack. In contrast to the original specification, we leave

open how the intruder process P exactly works, and rather specify that it is some process generated from the initial knowledge $K = \{in\ k, k', k''\}$:

$$
\begin{array}{ll}
(\mathit{Firewall} \mid \boxed{K}, \mathit{true}) & \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k[\mathit{in}\ k'.\mathit{in}\ w] \mid \boxed{K}, \mathit{true}) & \text{by rule (2)} \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k'[k[\mathit{in}\ w] \mid \boxed{K}] \mid \boxed{K}, \phi_1) & \text{by rule (7)} \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k'[\mathit{in}\ w \mid \boxed{K}] \mid \boxed{K}, \phi_2) & \text{by rule (11)} \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle \mid k'[\boxed{K}]] \mid \boxed{K}, \phi_2) & \text{by rule (1)} \\
\Rightarrow (w[\mathit{open}\ k''.\langle s\rangle \mid \boxed{K}] \mid \boxed{K}, \phi_2) & \text{by rule (3)} \\
\Rightarrow (w[\langle s\rangle \mid \boxed{K}] \mid \boxed{K}, \phi_3) & \text{by rule (12)} \\
\Rightarrow (w[\boxed{K \cup \{s\}}] \mid \boxed{K}, \phi_3) & \text{by rule (13)}
\end{array}
$$

where we have collected the constraints $\phi_1 = K \vdash k'$, $\phi_2 = \phi_1 \wedge K \vdash \mathit{open}\ w$, and $\phi_3 = \phi_2 \wedge K \vdash k''$. These constraints are satisfiable. This corresponds to the attack we had described on the ground model, only here we found it *lazily* during the search, rather than specifying the process up front. Another difference to the original trace is that we have an intruder process $\boxed{K}$ remaining at the outermost level the entire time. This reflects that the intruder process could be a parallel composition of two parts only one of which enters the firewall—the position outside the does not have to be "given up" by the intruder.

In the case that learning the secret s alone is not the goal, but to get it out of the firewall. Indeed we can apply rule (11) to the last reached state to get ($\boxed{K \cup \{s\}} \mid \boxed{K}, K \vdash \mathit{open}\ w \wedge \phi_3$). (Further, using the (15) rule, the two intruder processes can merge again, yielding $\boxed{K \cup \{s\}}$.) The new constraint $K \vdash \mathit{open}\ w$ however is not satisfiable, so this symbolic state has an empty semantics (no attack is realizable in this way) and can be discarded from the search. In fact, there is no reachable symbolic state with satisfiable constraints where the secret s is in an intruder process that is not below $w[\cdot]$.

Ambient in the Middle The previous example has basically identified how an honest client (authenticating itself by the knowledge of the keys k, k', and k'') is supposed to behave, namely $\mathit{Client} \equiv k'[\mathit{open}\ k.k''[C_0]]$ for some process C_0. We now consider the case that such an honest client and firewall execute in the presence of an intruder process K:

$$
\begin{array}{l}
(\mathit{Firewall} \mid \mathit{Client} \mid \boxed{K}, \mathit{true}) \\
\Rightarrow (\mathit{Firewall} \mid k'[\mathit{open}\ k.k''[C_0] \mid x[\boxed{K}]] \mid \boxed{K}, \phi_1) \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k[\mathit{in}\ k'.\mathit{in}\ w] \mid k'[\mathit{open}\ k.k''[C_0] \mid x[\boxed{K}]] \mid \boxed{K}, \phi_1) \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k'[k[\mathit{in}\ w] \mid \mathit{open}\ k.k''[C_0] \mid x[\boxed{K}]] \mid \boxed{K}, \phi_1) \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle] \mid k'[\mathit{in}\ w \mid k''[C_0] \mid x[\boxed{K}]] \mid \boxed{K}, \phi_1) \\
\Rightarrow (w[\mathit{open}\ k'.\mathit{open}\ k''.\langle s\rangle \mid k'[k''[C_0] \mid x[\boxed{K}]]] \mid \boxed{K}, \phi_1) \\
\Rightarrow (w[\mathit{open}\ k''.\langle s\rangle \mid k''[C_0] \mid x[\boxed{K}]] \mid \boxed{K}, \phi_1) \\
\Rightarrow (w[\langle s\rangle \mid k''[C_0] \mid \boxed{K}] \mid \boxed{K}, \phi_2)
\end{array}
$$

where $\phi_1 = K \vdash in\ k' \wedge K \vdash x$ and $\phi_2 = K \vdash in\ k' \wedge K \vdash k''$. Note that in the one but last step we apply the open action to the intruder ambient $x[\boxed{K}]$ (unifying $x = k_0$). Thus, the intruder can inject code into the firewall (without being isolated by $x[\cdot]$, and so he can obtain s) if he knows only $in\ k'$ and k''. The *open* k capability is not needed, since this is done by the client after the intruder has infected it.

A Communication Example As an example where capabilities are communicated consider the process $n_1[\boxed{K_1} \mid n_2[in\ n_3.\langle in\ n_4\rangle]] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]]$ where $K_1 = \{n_3, open\ n_2\}$ and $K_2 = \{open\ n_1\}$. Let the goal be that there is no intruder process who will know both *open* n_1 and *open* n_2. The lazy mobile ambient technique finds an attack as follows:

$$
\begin{array}{ll}
(n_1[\boxed{K_1} \mid n_3[\boxed{K_1} \mid n_2[\langle in\ n_4\rangle]]] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], true) & \\
\Rightarrow (n_1[\boxed{K_1} \mid n_3[\boxed{K_1} \mid \langle in\ n_4\rangle]] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], \phi_1) & \text{by rule (11)} \\
\Rightarrow (n_1[\boxed{K_1} \mid n_3[\boxed{K_1 \cup \{in\ n_4\}}]] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], \phi_1) & \text{by rule (13)} \\
\Rightarrow (n_1[\boxed{K_1} \mid \boxed{K_1 \cup \{in\ n_4\}}] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], \phi_2) & \text{by rule (11)} \\
\Rightarrow (n_1[\boxed{K_1 \cup \{in\ n_4\}}] \mid n_5[n_4[\boxed{K_2} \mid \langle out\ n_5\rangle]], \phi_2) & \text{by rule (15)} \\
\Rightarrow (n_1[\boxed{K_1 \cup \{in\ n_4\}}] \mid n_4[\boxed{K_2}] \mid n_5[0], \phi_2) & \text{by rule (2)} \\
\Rightarrow (n_4[\boxed{K_2} \mid n_1[\boxed{K_1 \cup \{in\ n_4\}}]] \mid n_5[0], \phi_3) & \text{by rule (6)} \\
\Rightarrow (n_4[\boxed{K_2} \mid \boxed{K_1 \cup \{in\ n_4\}}] \mid n_5[0], \phi_4) & \text{by rule (11)} \\
\Rightarrow (n_4[\boxed{K_2 \cup K_1 \cup \{in\ n_4\}}] \mid n_5[0], \phi_4) & \text{by rule (15)}
\end{array}
$$

where we the following satisfiable constraints: $\phi_1 = K_1 \vdash open\ n_2$, $\phi_2 = \phi_1 \wedge K_1 \vdash open\ n_3$, $\phi_3 = \phi_2 \wedge K_1 \cup \{in\ m_4\} \vdash in\ m_4$, and $\phi_4 = \phi_3 \wedge K_2 \vdash open\ n_1$. We have reached a state where an intruder process knows both *open* n_1 and *open* n_2.

4 Conclusions

We have transferred the symbolic lazy intruder technique from protocol verification to a different problem: an intruder who creates malicious code for execution on some honest platform. This gives us an efficient method to check whether the platform achieves its security goals for *any* intruder code, because we avoid the naive search of the space of possible programs that the intruder can come up with. Instead we determine this code in a demand-driven, lazy way.

Our approach is closest to a model-checking technique. In contrast to static analysis approaches, it works without over-approximation, but requires a bounding of the number of steps that honest agents can perform. The symbolic nature however allows to work without any bound on the size of programs that the intruder can generate. This is similar to the original use of the lazy intruder in protocol verification [13, 15, 18, 7].

We have used a fragment of the mobile ambient calculus with communication as a small and succinct formalism to model both the platform and the mobile

code [9]. We have omitted the replication operator in order to bound honest processes (though not the intruder). We have omitted the path constraints because they induce considerable complications for our approach and leave their integration for future work. We also plan to consider the extension of boxed ambients introduced by Bugliesi et al. [8] which add interesting means for access control and communication. Moreover it is possible to extend the ambient calculus and our method to support cryptographic operators (like encryption and signing) in the communication of processes.

We believe that the approach we have presented here is generally applicable to the formal analysis of platforms that host mobile code. The key elements can be summarized as follows. First, the code can be lazily developed by exploring at each step which operations can be performed next and what data is needed. This data is handled lazily as well. Second, the intruder code has a notion of knowledge that it can use in further operations and communications, and every received message adds to this knowledge. Third, the code may be able to move to other locations; two pieces of intruder code that meet then pool their knowledge.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *ACM symposium on principles of programming languages*, pages 104–115, 2001.
2. M. Abadi and C. Fournet. Private Authentication. *Theoretical Computer Science*, 322(3):427 – 476, 2004.
3. J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *IEEE Symposium on Security and Privacy*, pages 2–11, 2001.
4. M. Arapinis and M. Duflot. Bounding messages for free in security protocols. In *FSTTCS*, pages 376–387, 2007.
5. T. Avanesov, Y. Chevalier, M. Rusinowitch, and M. Turuani. Intruder deducibility constraints with negation. *CoRR*, abs/1207.4871, 2012.
6. F. Baader. Unification in commutative theories, hilbert's basis theorem, and gröbner bases. *J. ACM*, 40(3):477–503, 1993.
7. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
8. M. Bugliesi, G. Castagna, and S. Crafa. Access control for mobile agents: The calculus of boxed ambients. *ACM Trans. Program. Lang. Syst.*, 26(1):57–124, 2004.
9. L. Cardelli and A. D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.
10. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the Security of Protocols with Diffie-Hellman Exponentiation and Products in Exponents. In *FST TCS'03*, LNCS 2914, pages 124–135, 2003.
11. S. Delaune, P. Lafourcade, D. Lugiez, and R. Treinen. Symbolic protocol analysis for monoidal equational theories. *Inf. Comput.*, 206(2-4):312–351, 2008.
12. T. Groß, B. Pfitzmann, and A.-R. Sadeghi. Browser model for security analysis of browser-based protocols. In *ESORICS*, pages 489–508, 2005.
13. A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. FLOC'99 Workshop on Formal Methods and Security Protocols*, 1999.

14. A. Kassem, P. Lafourcade, Y. Lakhnech, and S. Mödersheim. Multiple independent lazy intruders. In *HotSpot 2013*, 2013. To Appear.
15. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS'01*, pages 166–175. ACM Press, 2001.
16. S. Mödersheim, F. Nielson, and H. R. Nielson. Lazy mobile intruders (extended version). Technical Report IMM-TR-2012-13, DTU Informatics, 2012. imm.dtu.dk/ samo.
17. G. C. Necula. Proof-carrying code. In *POPL*, pages 106–119, 1997.
18. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions, composed keys is NP-complete. *Theor. Comput. Sci.*, 1-3(299):451–475, 2003.