Basic Research in Computer Science

Recursion vs. Replication in Simple Cryptographic Protocols

Hans Hüttel
Jiří Srba

BRICS Report Series **RS-04-23**

ISSN 0909-0878 **October 2004**

See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/04/23/`

Recursion vs. Replication in Simple Cryptographic Protocols

Hans Hüttel* and Jiří Srba**

BRICS* ***, Department of Computer Science, University of Aalborg
Fredrik Bajersvej 7B, 9220 Aalborg East, Denmark

Abstract. We use some recent techniques from process algebra to draw several conclusions about the well studied class of ping-pong protocols introduced by Dolev and Yao. In particular we show that all nontrivial properties, including reachability and equivalence checking wrt. the whole van Glabbeek's spectrum, become undecidable for a very simple recursive extension of the protocol. The result holds even if no nondeterministic choice operator is allowed. We also show that the extended calculus is capable of an implicit description of the active intruder, including full analysis and synthesis of messages in the sense of Amadio, Lugiez and Vanackère. We conclude by showing that reachability analysis for a replicative variant of the protocol becomes decidable.

1 Introduction

Process calculi have been suggested as a natural vehicle for reasoning about cryptographic protocols. In [1], Abadi and Gordon introduced the spi-calculus (a variant of the π-calculus) and described how properties such as secrecy and authenticity can be expressed via notions of observational equivalence (like may-testing). Alternatively, security questions have been studied using reachability analysis [3, 5, 11].

We provide a basic study of expressiveness and feasibility of cryptographic protocols. We are interested in two verification approaches: *reachability analysis* and *equivalence (preorder) checking*. In reachability analysis the question is whether a certain (bad or good) configuration of the protocol is reachable from a given initial one. In equivalence checking the question is whether a protocol implementation is equivalent (e.g. bisimilar) to a given specification (optimal behaviour). These verification strategies can be used even in the presence of an *active intruder* (in the Dolev-Yao style), i.e., an agent with capabilities to listen to any communication, to perform analysis and synthesis of communicated messages according to the actual knowledge of compromised keys, and to actively

* hans@cs.auc.dk

** srba@cs.auc.dk, the author is supported in part by the GACR, grant No. 201/03/1161.

*** **B**asic **R**esearch in **C**omputer **S**cience,
Centre of the Danish National Research Foundation.

participate in the protocol behaviour by transmitting new messages. This can be naturally implemented not only into the reachability analysis (see e.g. [4]) but also into the equivalence checking approach. As described in [12], these questions for equivalence (preorder) checking approach can be formulated as follows: "a protocol P guarantees a security property X if, whatever hostile environment E with a certain initial knowledge ϕ_I, then P is equivalent (in preorder) to (with) the specification $\alpha(P)$." Formally this is given by saying that

$$\text{protocol P satisfies property } X \quad \text{iff} \quad \forall E \in \mathcal{E} : \; P \parallel E \approx \alpha(P). \tag{1}$$

By an appropriate choice of the specification function α and a suitable equivalence (preorder) $\approx$, several security properties can be verified. Here is a small selection:

- *Secrecy* (confidential information should be available only to the partners of the communication). Here $\approx$ stands for trace preorder.
- *(Message) authenticity* (identification of other agents (messages) participating in communication). Here $\approx$ stands for trace equivalence or preorder.
- *Fairness* (in a contract, no party can gain advantage by ending the protocol prematurely). Here $\approx$ stands for failure equivalence.

Various notions of bisimilarity are studied in this context as bisimilarity is usually the "most decidable behavioral equivalence" as confirmed e.g. by several positive decidability results in process algebra [6]. Hence the questions whether a certain class of cryptographic protocols has decidable reachability and equivalence (bisimilarity) checking are of particular importance for automated verification.

A number of security properties are decidable for finite protocols [3, 19]. In the case of an unbounded number of protocol configurations, the picture is more complex. Durgin et al. showed in [10] that security properties are undecidable in a restricted class of so-called bounded protocols (that still allows for infinitely many reachable configurations). In [2] Amadio and Charatonik consider a language of tail-recursive protocols with bounded encryption depth and name generation; they show that, whenever certain restrictions on decryption are violated, one can encode two-counter machines in the process language. On the other hand, Amadio, Lugiez and Vanackère show in [4] that the reachability problem is in PTIME for a class of protocols with iteration.

In this paper we focus solely on ping-pong based behaviours of recursive and replicative protocols (perhaps the simplest behaviour of all studied calculi) in order to draw general conclusions about expressiveness and tractability of formal verification of cryptographic protocols. The class of *ping-pong protocols* was introduced in 1983 by Dolev and Yao [9]. The formalism deals with memory-less protocols which may be subjected to arbitrarily long attacks. Here, the secrecy of a finite ping-pong protocol can be decided in polynomial time. Later, Dolev, Even and Karp found a cubic-time algorithm [8]. The class of protocols studied in [4] contains iterative ping-pong protocols and, as a consequence, secrecy properties remain polynomially decidable even in this case.

In the present paper we continue our study of recursive and replicative extensions of ping-pong protocols. In [14] we showed that the recursive extension of the calculus is Turing powerful, however, the nondeterministic choice operator appeared to be essential in the construction. The question whether the calculus is Turing powerful even without any explicit way to define nondeterministic processes was left open. Here we present a radically new reduction from multi-stack automata and strengthen the undecidability results to hold even for protocols without nondeterministic choice. We prove, in particular, that both reachability and equivalence checking for all equivalences and preorders between trace equivalence/preorder and isomorphism of labelled transition systems (which includes all equivalences and preorders from van Glabbeek's spectrum [20]) become undecidable. These results are of general importance because they prove the impossibility of automated verification for essentially all recursive cryptographic protocols capable of at least the ping-pong behaviour.

In the initial study from [14], the question of active attacks on the protocol was not dealt with. We shall demonstrate that a complete notion of the active intruder (including analysis and synthesis of messages in the sense of Amadio, Lugiez and Vanackère [4]) can be explicitly encoded into our formalism in order to analyze general properties like in the scheme (1).

Finally, we study a replicative variant of the calculus. Surprisingly, such a calculus becomes decidable, at least with regard to reachability analysis. We use a very recent result from process algebra (decidability of reachability for weak process rewrite systems by Křetínský, Řehák and Strejček [15]) in order to derive the result. We believe that this is one of the reasons which formally confirm the general trend that replication is a good choice for cryptographic formalisms and that is why recursion is only rarely studied.

2 Basic definitions

2.1 Labelled transition systems with label abstraction

In order to provide a uniform framework for our study of ping-pong protocols, we define their semantics by means of labelled transition systems. A *labelled transition system* (LTS) is a triple $\mathcal{T} = (S, \mathcal{A}ct, \longrightarrow)$ where S is a set of *states* (or *processes*), $\mathcal{A}ct$ is a set of *labels* (or *actions*), and $\longrightarrow \subseteq S \times \mathcal{A}ct \times S$ is a *transition relation*, written $\alpha \xrightarrow{a} \beta$, for $(\alpha, a, \beta) \in \longrightarrow$. As usual we extend the transition relation to the elements of $\mathcal{A}ct^*$. We also write $\alpha \longrightarrow^* \beta$, whenever $\alpha \xrightarrow{w} \beta$ for some $w \in \mathcal{A}ct^*$.

The idea is that the states represent *global configurations* of a given protocol and the transitions describe the *information flow*. Labels on the transitions moreover represent the messages (both plain-text and cipher-text) which are being communicated during the state changes.

Remark 1. In [14] the semantics of ping-pong protocols is given in terms of transition systems with knowledge, i.e., unlabelled transition systems where each state it assigned its knowledge, represented as a subset of a certain set of all

possible knowledge values. By standard techniques such a knowledge-based semantics can be translated to labelled transition systems and the studied verification properties (reachability, equivalence checking, etc.) are preserved. For example a state A with two knowledge values p_1 and p_2 can be transformed to a labelled transition system where the values p_1 and p_1 are represented as self-loops in state A which are visible under special actions p_1 and p_2. A fresh action a is used to represent the change of the state (the unlabelled transitions in the original knowledge-based semantics).

The explicit possibility to observe the full content of messages is sometimes not very realistic; it means that an external observer of such a system can e.g. distinguish between two different messages encrypted by the same encryption key, without the actual knowledge of the key.

In order to restrict capabilities of the observer we introduce a so called *label abstraction function* $\phi : \mathcal{A}ct \mapsto \mathcal{A}ct$. Given a LTS $\mathcal{T} = (S, \mathcal{A}ct, \longrightarrow_{\mathcal{T}})$ and a label abstraction function ϕ we define a new LTS $\mathcal{T}_\phi \stackrel{\text{def}}{=} (S, \mathcal{A}ct, \longrightarrow_{\mathcal{T}_\phi})$ where $\alpha \stackrel{\phi(a)}{\longrightarrow}_{\mathcal{T}_\phi} \beta$ iff $\alpha \stackrel{a}{\longrightarrow}_{\mathcal{T}} \beta$ for all $\alpha, \beta \in S$ and $a \in \mathcal{A}ct$. We call $\mathcal{T}_\phi$ a *labelled transition system with label abstraction.*

Let us now focus on the messages (actions). Assume a given set of encryption keys $\mathcal{K}$. The set of all messages over $\mathcal{K}$ is given by the following abstract syntax

$$m \ ::= \ k \ \mid \ k \cdot m$$

where k ranges over $\mathcal{K}$. Hence every element of the set $\mathcal{K}$ is a *(plain-text) message* and if m is a message then $k \cdot m$ is a *(cipher-text) message* (meaning that the message m is encrypted by the key k). Given a message $k_1 \cdot k_2 \cdots k_n$ over $\mathcal{K}$ we usually[1] write it only as a word $k_1 k_2 \cdots k_n$ from $\mathcal{K}^*$. Note that k_n is the plain-text part of the message and the outermost encryption key is always on the left (k_1 in our case). In what follows we shall identify the set of messages and $\mathcal{K}^*$, and we denote the extra element of $\mathcal{K}^*$ consisting of the empty sequence of keys by ϵ.

Example 1. Let us consider a labelled transition system $\mathcal{T} \stackrel{\text{def}}{=} (S, \mathcal{A}ct, \longrightarrow)$ where $S \stackrel{\text{def}}{=} \{A, B, C\}$, $\mathcal{A}ct \stackrel{\text{def}}{=} \mathcal{K}^*$ for a given set of keys $\mathcal{K} = \{k_1, k_2, \lambda\}$ and $\longrightarrow$ is given by the following picture.

$$A \xrightarrow{k_1 k_2} B \xrightarrow{k_2} C$$

The protocol computation starts in the state A and is very simple. First a plain-text k_2 encrypted by the encryption key k_1 is communicated to the process B, which decrypts the message and sends out the plain-text k_2. Let us now assume a label abstraction function ϕ defined by $\phi(k) = k$ if $k \in \mathcal{K}$ and $\phi(m) = \lambda$

[1] In our previous work on ping-pong protocols [14] we denoted a message m encrypted by a key k as $\{m\}_k$. We changed the notation in order to improve the clarity of the proofs. In particular, when messages like $k_1 k_2 \cdots k_n$ are used, the previous syntax described the keys in a reversed order, which was technically inconvenient.

otherwise. The labelled transition system T_ϕ with label abstraction function ϕ now looks as follows.

$$A \xrightarrow{\lambda} B \xrightarrow{k_2} C$$

This translates to the fact that the external observer is not allowed to see the content of encrypted messages (the action λ is used instead) and only plain-text messages can be recognized. □

The level of abstraction we may select depends on the particular studied property we are interested in and it directly corresponds to the specification function α from (1). Nevertheless, it seems reasonable to require at least the possibility to distinguish between plain-text and cipher-text messages. We say that a label abstraction function ϕ is *reasonable* iff $\phi(k) \neq \phi(k'w)$ for all $k, k' \in \mathcal{K}$ and $w \in \mathcal{K}^+$.

2.2 A calculus of recursive ping-pong protocols

We shall now define a calculus which captures exactly the class of ping-pong protocols by Dolev and Yao [9] extended (in a straightforward manner) with recursive definitions.

Let $\mathcal{K}$ be a set of encryption keys. A *specification* of a recursive ping-pong is a finite set of process definitions Δ such that for every *process constant* P (from a given set $\mathcal{C}onst$) the set Δ contains exactly one process definition of the form

$$P \stackrel{\text{def}}{=} \sum_{i_1 \in I_1} v_{i_1} \rhd . \overline{w_{i_1} \rhd}.P_{i_1} + \sum_{i_2 \in I_2} v_{i_2}.P_{i_2} + \sum_{i_3 \in I_3} \overline{w_{i_3}}.P_{i_3}$$

where I_1, I_2 and I_3 are finite sets of indices such that $I_1 \cup I_2 \cup I_3 \neq \emptyset$, and v_{i_1}, v_{i_2}, w_{i_1} and w_{i_3} are messages (belong to $\mathcal{K}^*$) for all $i_1 \in I_1$, $i_2 \in I_2$ and $i_3 \in I_3$, and $P_i \in \mathcal{C}onst \cup \{\mathbf{0}\}$ for all $i \in I_1 \cup I_2 \cup I_3$ such that $\mathbf{0}$ is a special constant called the *empty process*. We moreover require that v_{i_2} and w_{i_3} for all $i_2 \in I_2$ and $i_3 \in I_3$ are different from the empty message ϵ. (Observe that any specification Δ contains only finitely many keys.)

Summands continuing in the empty process constant $\mathbf{0}$ will be written without the $\mathbf{0}$ symbol and process definitions will often be written in their unfolded form using the *nondeterministic choice operator* '+'. An example of a process definition is e.g. $P \stackrel{\text{def}}{=} k_1 \rhd . \overline{k_2 \rhd}.P_1 + k_1 \rhd . \overline{k_3 \rhd} + k_1 k_2.P_1 + \overline{k_1 k_1} + \overline{k_1 k_2}.P_2$.

The intuition is that each summand of the form $v_{i_1} \rhd . \overline{w_{i_1} \rhd}.P_{i_1}$ can receive a message encrypted by a sequence v_{i_1} of outermost keys, decrypt the message using these keys, send it out encrypted by the sequence of keys w_{i_1}, and finally behave as the process constant P_{i_1}. The symbol $\rhd$ stands for the rest of the message after decrypting it with the key sequence v_{i_1}. This describes a standard ping-pong behaviour of the process.

In addition to this we may have summands of the forms $v_{i_2}.P_{i_2}$ and $\overline{w_{i_3}}.P_{i_3}$, meaning simply that a message is received and forgotten or unconditionally transmitted, respectively. This is a small addition to the calculus we presented

in [14] in order to allow for discarding of old messages and generation of new messages. These two features were not available in the earlier version of the calculus but they appear to be technically convenient when modeling an explicit intruder and for strengthening the positive decidability results in Section 5. Nevertheless, the undecidability results presented in Section 3 are valid even without this extension since only the standard ping-pong behaviour is used in the constructions. A feature very similar to the forgetful input operation can be also found in [4].

A *configuration* of a ping-pong protocol specification Δ is a parallel composition of process constants, possibly preceded by output messages. Formally the set $\mathcal{C}onf$ of configurations is given by the following abstract syntax

$$C ::= \quad \mathbf{0} \quad | \quad P \quad | \quad \overline{w}.P \quad | \quad C \parallel C$$

where $\mathbf{0}$ is the empty configuration, $P \in \mathcal{C}onst \cup \{\mathbf{0}\}$ ranges over process constants including the empty process, $w \in \mathcal{K}^*$ ranges over the set of messages, and '$\parallel$' is the operator of parallel composition.

We introduce a structural congruence relation $\equiv$ which identifies configurations that represent the same state of the protocol. The relation $\equiv$ is defined as the least congruence over configurations ($\equiv \subseteq \mathcal{C}onf \times \mathcal{C}onf$) such that $(\mathcal{C}onf, \parallel, \mathbf{0})$ is a commutative monoid and $\overline{\epsilon}.P \equiv P$ for all $P \in \mathcal{C}onst$. In what follows we shall identify configurations up to structural congruence.

Remark 2. We let $\overline{\epsilon}.P \equiv P$ because the empty message should never be communicated. This means that when a prefix like $k \triangleright \,.\, \overline{\triangleright}.P$ receives a plain-text message k and tries to output $\overline{\epsilon}.P$, it simply continues as the process P.

We shall now define the semantics of ping-pong protocols in terms of labelled transition systems. We define a set $\mathcal{C}onf_S \subseteq \mathcal{C}onf$ consisting of all configurations that do not contain the operator of parallel composition and call these *simple configurations*. We also define two sets $In(C, m), Out(C, m) \subseteq \mathcal{C}onf_S$ for all $C \in \mathcal{C}onf_S$ and $m \in \mathcal{K}^+$. The intuition is that $In(C, m)$ ($Out(C, m)$) contains all configurations which can be reached from the simple configuration C after receiving (resp. outputting) the message m from (to) the environment. Formally, $In(C, m)$ and $Out(C, m)$ are the smallest sets which satisfy:

- $Q \in In(P, m)$ whenever $P \in \mathcal{C}onst$ and $m.Q$ is a summand of P
- $\overline{w\alpha}.Q \in In(P, m)$ whenever $P \in \mathcal{C}onst$ and $v \triangleright \,.\, \overline{w \triangleright}.Q$ is a summand of P such that $m = v\alpha$
- $P \in Out(\overline{m}.P, m)$ whenever $P \in \mathcal{C}onst \cup \{\mathbf{0}\}$
- $Q \in Out(P, m)$ whenever $P \in \mathcal{C}onst$ and $\overline{m}.Q$ is a summand of P.

A given protocol specification Δ determines a labelled transition system $T(\Delta) \stackrel{\text{def}}{=} (S, \mathcal{A}ct, \longrightarrow)$ where the states are configurations of the protocol modulo the structural congruence ($S \stackrel{\text{def}}{=} \mathcal{C}onf/\equiv$), the set of labels (actions) is the set of messages that can be communicated between the agents of the protocol ($\mathcal{A}ct \stackrel{\text{def}}{=} \mathcal{K}^+$), and the transition relation $\longrightarrow$ is given by the following SOS rule (recall that '$\parallel$' is commutative).

$$\frac{m \in \mathcal{K}^+ \qquad C_1, C_2 \in \mathcal{C}onf_S \qquad C_1' \in \mathit{Out}(C_1, m) \qquad C_2' \in \mathit{In}(C_2, m)}{C_1 \parallel C_2 \parallel C \xrightarrow{m} C_1' \parallel C_2' \parallel C}$$

This means that (in the context C) two simple configurations (agents) C_1 and C_2 can communicate a message m in such a way that C_1 outputs m and becomes C_1' while C_2 receives the message m and becomes C_2'.

Example 2. Let us consider a protocol specification Δ.

$$P \stackrel{\text{def}}{=} \overline{k}.P \;\parallel\; k\triangleright\,.\,\overline{kk\triangleright}.P \;\parallel\; k.Q \qquad\qquad Q \stackrel{\text{def}}{=} k.P$$

A fragment of the labelled transition system reachable from the initial configuration $P \parallel P$ looks as follows.

$$\begin{array}{ccccccc} P \parallel P & \xrightarrow{\quad k \quad} & P \parallel \overline{kk}.P & \xrightarrow{\quad kk \quad} & P \parallel \overline{kkk}.P & \xrightarrow{\quad kkk \quad} & \cdots \\ {\scriptstyle k}\big(\quad \big){\scriptstyle k} & & & & & & \\ P \parallel Q & & & & & & \end{array}$$

□

For further discussion and examples of recursive ping-pong protocols we refer the reader to [14].

2.3 Reachability and behavioural equivalences

One of the problems that is usually studied is that of *reachability analysis*: given two configurations $C_1, C_2 \in \mathcal{C}onf$ we ask whether C_2 is reachable from C_1, i.e., if $C_1 \longrightarrow^* C_2$. In this case the set of labels is irrelevant.

As the semantics of our calculus is given in terms of labelled transition systems (together with an appropriate label abstraction function), we can also study the *equivalence checking* problems. Given some behavioural equivalence or preorder $\leftrightarrow$ from van Glabbeek's spectrum [20] (e.g. strong bisimilarity or trace, failure and simulation equivalences/preorders just to mention a few) and two configurations $C_1, C_2 \in \mathcal{C}onf$ of a protocol specification Δ, the question is to decide whether C_1 and C_2 are $\leftrightarrow$-equivalent (or $\leftrightarrow$-preorder related) in $T(\Delta)$, i.e., whether $C_1 \leftrightarrow C_2$.

3 Recursive ping-pong protocols without explicit choice

In this section we strengthen the undecidability result from [14] and show that the reachability and equivalence checking problems are undecidable for ping-pong protocols without an explicit operator of nondeterminism and using classical ping-pong behaviour only, i.e., for protocols without any occurrence of the choice operator '+' and where every defining equation is of the form $P \stackrel{\text{def}}{=} v\triangleright\,.\,\overline{w\triangleright}.P'$ such that $P' \in \mathcal{C}onst$.

Remark 3. Note that every process constant is allowed to have exactly one defining equation, however, no constraints are imposed on the communication behaviour of the parallel components.

We moreover show that the negative results apply to all behavioural equivalences and preorders between trace equivalence/preorder and isomorphism of LTS (which preserves labelling) with regard to all reasonable label abstraction functions as defined in Section 2.

These results are achieved by showing that recursive ping-pong protocols can step-by-step simulate a Turing powerful computational device, in our case a computational model called multi-stack machines.

A *multi-stack machine* R with ℓ stacks ($\ell \geq 1$) is a triple $R = (Q, \Gamma, \longrightarrow)$ where Q is a finite set of *control-states*, Γ is a finite *stack alphabet* such that $Q \cap \Gamma = \emptyset$, and $\longrightarrow \subseteq Q \times \Gamma \times Q \times \Gamma^*$ is a finite set of *transition rules*, written $pX \longrightarrow q\alpha$ for $(p, X, q, \alpha) \in \longrightarrow$.

A *configuration* of a multi-stack machine R is an element from $Q \times (\Gamma^*)^\ell$. We assume a given initial configuration $(q_0, w_1, \ldots, w_\ell)$ where $q_0 \in Q$ and $w_i \in \Gamma^*$ for all i, $1 \leq i \leq \ell$. If some of the stacks w_i are empty, we denote them by ϵ.

A *computational step* is defined such that whenever there is a transition rule $pX \longrightarrow q\alpha$ then a configuration which is in the control-state p and has X on top of the i'th stack (the tops of the stacks are on the left) can perform the following transition:

$$(p, w_1, \ldots, Xw_i, \ldots, w_\ell) \longrightarrow (q, w_1, \ldots, \alpha w_i, \ldots, w_\ell)$$

for all $w_1, \ldots, w_\ell \in \Gamma^*$ and for all i, $1 \leq i \leq \ell$.

It is a folklore result that multi-stack machines are Turing powerful. Hence (in particular) the following problem is easily seen to be undecidable: given an initial configuration $(q_0, w_1, \ldots, w_\ell)$ of a multi-stack machine R, can we reach the configuration $(h, \epsilon, \ldots, \epsilon)$ for a distinguished halting control-state $h \in Q$ such that all stacks are empty? Without loss of generality we can even assume that a configuration in the control-state h is reachable iff all stacks are empty.

Let $R = (Q, \Gamma, \longrightarrow)$ be a multi-stack machine. We define the following set of keys of a ping-pong specification Δ: $\mathcal{K} \stackrel{\text{def}}{=} Q \cup \Gamma \cup \{k_p \mid p \in Q\} \cup \{t, k_*\}$. Here t is a special key such that every communicated message is an encryption of the plain-text key t. The reason for this is that it ensures that the protocol never communicates any plain-text message. The key k_* is a special purpose locking key and it is explained later on in the construction.

We shall construct a ping-pong protocol specification Δ as follows.

- For every transition rule $pX \longrightarrow q\alpha$ we have a process constant $P_{pX \longrightarrow q\alpha}$ with the following defining equation.

$$P_{pX \longrightarrow q\alpha} \stackrel{\text{def}}{=} pX\rhd \,.\, \overline{k_q \alpha \rhd}.P_{pX \longrightarrow q\alpha}$$

- For every state $p \in Q$ we have two process constants T_p and T'_p.

$$T_p \stackrel{\text{def}}{=} k_p\rhd \,.\, \overline{k_*\rhd}.T'_p$$

$$T'_p \stackrel{\text{def}}{=} k_* \rhd \,.\, \overline{p \rhd}.T_p \quad \text{if } p \in Q \setminus \{h\}, \text{ and} \quad T'_h \stackrel{\text{def}}{=} h \rhd \,.\, \overline{h \rhd}.T'_h$$

Recall that $h \in Q$ is the halting control-state.

– Finally, we define a process constant B (standing for a buffer over a fixed key k_*).

$$B \stackrel{\text{def}}{=} k_* \rhd \,.\, \overline{k_* \rhd}.B$$

In this defining equation the key k_* locks the content of the buffer such that it is accessible only by some T'_p.

Note that Δ does not contain any choice operator '+' as required.

Let $(q_0, w_1, \ldots, w_\ell)$ be an initial configuration of the multi-stack machine R. The corresponding initial configuration of the protocol Δ is defined as follows (the meta-symbol Π stands for a parallel composition of the appropriate components).

$$\Big(\prod_{(r,A,s,\beta) \in \longrightarrow} P_{rA \longrightarrow s\beta} \Big) \;\|\; \Big(\prod_{p \in Q \setminus \{q_0\}} T_p \Big) \;\|\; T'_{q_0} \;\|\; \Big(\prod_{j \in \{1,\ldots,\ell\}} \overline{k_* w_j t}.B \Big) \qquad (2)$$

The following invariants will be preserved during any computational sequence starting from this initial configuration:

– at most one T'_p for some $p \in Q$ is present as a parallel component (the intuition is that this represents the fact that the machine R is in the control-state p), and
– plain-text messages are never communicated.

Let $(p, w_1, \ldots, w_i, \ldots w_\ell) \longrightarrow (q, w_1, \ldots, \alpha w'_i, \ldots w_\ell)$ be a computational step of R using the rule $pX \longrightarrow q\alpha$ such that $w_i = Xw'_i$. This one step is simulated by a sequence of four transitions in the ping-pong protocol Δ (see Figure 3). In the first step one buffer is selected and unlocked (the current control-state p replaces the locking key k_* in the outermost encryption). In particular the buffer $\overline{k_* w_i t}.B$ can be unlocked. No other kinds of transitions are possible in the first step. Opening of the selected buffer means that some of the process constants $P_{rA \longrightarrow s\beta}$ become able to accept this message. In particular the process constant $P_{pX \longrightarrow q\alpha}$ can receive the message and output $\overline{k_q \alpha w_i t}$ for further communication; the key k_q determines the control-state change. (At this stage also a communication between $\overline{k_* w_i t}.B$ and B is enabled but it does not change the current state and hence it cannot contribute to a computational progress.) In the next step only T_q can receive the message $\overline{k_q \alpha w_i t}$, it remembers the new control-state q by becoming T'_q and offers the k_*-locked message for a communication with B. This last communication (when B receives back the modified buffer) ends a simulation of one computational step of R.

The following property is easy to see: if after some number of steps starting in a protocol configuration corresponding to $(p, w_1, \ldots, w_\ell)$ we reach a first protocol configuration where T'_q appears for some $q \in Q$ then this corresponds to one correct computational step in R. On the other hand, the computation of Δ can

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \parallel \Big(\prod_{r\in Q\setminus\{p\}} T_r\Big) \parallel T'_p \parallel \Big(\prod_{j\in\{1,\ldots,\ell\}} \overline{k_* w_j t}.B\Big)$$

$$\downarrow k_* w_i t$$

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \parallel \Big(\prod_{r\in Q\setminus\{p\}} T_r\Big) \parallel \overline{pw_i t}.T_p \parallel \Big(\prod_{j\in\{1,\ldots,\ell\},j\neq i} \overline{k_* w_j t}.B\Big) \parallel B$$

$$\downarrow pw_i t$$

$$\Big(\prod_{(r,A,s,\beta)\in(\longrightarrow\setminus\{(p,X,q,\alpha)\})} P_{rA\longrightarrow s\beta}\Big) \parallel \overline{k_q \alpha w'_i t}.P_{pX\longrightarrow q\alpha} \parallel \Big(\prod_{r\in Q} T_r\Big) \parallel \Big(\prod_{j\in\{1,\ldots,\ell\},j\neq i} \overline{k_* w_j t}.B\Big) \parallel B$$

$$\downarrow k_q \alpha w'_i t$$

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \parallel \Big(\prod_{r\in Q\setminus\{q\}} T_r\Big) \parallel \overline{k_* \alpha w'_i t}.T'_q \parallel \Big(\prod_{j\in\{1,\ldots,\ell\},j\neq i} \overline{k_* w_j t}.B\Big) \parallel B$$

$$\downarrow k_* \alpha w'_i t$$

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \parallel \Big(\prod_{r\in Q\setminus\{q\}} T_r\Big) \parallel T'_q \parallel \Big(\prod_{j\in\{1,\ldots,\ell\},j\neq i} \overline{k_* w_j t}.B\Big) \parallel \overline{k_* \alpha w'_i t}.B$$

Fig. 1. Simulation of $(p, w_1, \ldots, w_i, \ldots w_\ell) \longrightarrow (q, w_1, \ldots, w'_i\alpha, \ldots w_\ell)$ s.t. $w_i = Xw'_i$

get stuck after the first communication step (in case that the unlocked buffer does not enable an application of any rule $rA \longrightarrow s\beta$) or an infinite sequence of communication steps of the form $\overline{m}.B \parallel B \xrightarrow{m} \overline{m}.B \parallel B$ is also possible.

This is formally captured in the following lemma.

Lemma 1. *In the given multi-stack machine R the configuration $(q, w'_1, \ldots, w'_\ell)$ is reachable from $(p, w_1, \ldots, w_\ell)$ if and only if*

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \parallel \Big(\prod_{p\in Q\setminus\{q\}} T_p\Big) \parallel T'_q \parallel \Big(\prod_{j\in\{1,\ldots,\ell\}} \overline{k_* w'_j t}.B\Big)$$

is reachable (in Δ) from

$$\Big(\prod_{(r,A,s,\beta)\in\longrightarrow} P_{rA\longrightarrow s\beta}\Big) \parallel \Big(\prod_{p\in Q\setminus\{p\}} T_p\Big) \parallel T'_p \parallel \Big(\prod_{j\in\{1,\ldots,\ell\}} \overline{k_* w_j t}.B\Big).$$

The following theorems are now easily derived.

Theorem 1. *The reachability problem for recursive ping-pong protocols without an explicit choice operator is undecidable.*

Proof. Immediately from Lemma 1 and from the undecidability of reachability for multi-stack machines. □

Theorem 2. *The equivalence checking problem for recursive ping-pong protocols without an explicit choice operator is undecidable for any behavioral equivalence/preorder between trace equivalence/preorder and isomorphism (including all equivalences and preorders from van Glabbeek's spectrum [20]) and for any reasonable label abstraction function.*

Proof. Let R be a multi-stack machine and Δ the protocol specification constructed above with the initial configuration C as given by (2). We consider the question whether $C \parallel \overline{h}$ is equivalent (or in preorder) with C.

In case that the halting control-state h is not reachable from the initial configuration of R, we know from Lemma 1 that T'_h will never appear as a parallel component in any reachable state from C. This implies that the plain-text message $\overline{h}$ will never be communicated and hence $C \parallel \overline{h}$ and C exhibit isomorphic behaviours under any label abstraction function.

On the other hand, if h is reachable from the initial configuration of R then because of Lemma 1 a configuration in Δ with the parallel component T'_h is reachable. Such a configuration is stuck in the process on the right, however, in the process on the left the plain-text message h can be communicated between T'_h and the extra parallel component $\overline{h}$. This means that $C \parallel \overline{h}$ and C are not even related by the trace preorder (and hence they are also not trace equivalent) because after a finite sequence of communicated messages there is a successor of the configuration $C \parallel \overline{h}$ which can communicate the plain-text h while (as argued before) C can only exchange cipher-text messages. As the label abstraction function ϕ is reasonable, necessarily for all messages m (cipher-texts) communicated in C it is the case that $\phi(m) \neq \phi(h)$.

To sum up, if the machine R cannot reach the halting configuration then $C \parallel \overline{h}$ and C are isomorphic and if R halts than $C \parallel \overline{h}$ and C are not in trace preorder. This implies that all equivalences and preorders between trace and isomorphism are undecidable for any reasonable label abstraction function. □

4 The active intruder

In the literature on applying process calculi to the study of cryptographic protocols, there have been several proposals for explicit modelling the active intruder (environment). Foccardi, Gorrieri and Martinelli in [12] express the environment within the process calculus, namely as a process running in parallel with the protocol. In [4] Amadio, Lugiez and Vanackère describe a tiny process calculus similar to ours, except that they use replication instead of recursion. Moreover, the environment is described in the semantics of the calculus. Transitions are of the form

$$(C, T) \rightarrow (C', T')$$

where C and C' are protocol configurations and T and T' denote the sets of messages known to the environment (all communication occurs only by passing messages through these sets).

The environment is assumed to be hostile; it may compute new messages by means of the operations of analysis and synthesis and pass these on to the process. Let $\mathcal{K}$ be a set of encryption keys as before. The *analysis* of a set of messages $T \subseteq \mathcal{K}^*$ is the least set $\mathcal{A}(T)$ satisfying

$$\mathcal{A}(T) = T \cup \{w \mid kw \in \mathcal{A}(T),\ k \in \mathcal{K} \cap \mathcal{A}(T)\}. \tag{3}$$

The *synthesis* of a set of messages $T \subseteq \mathcal{K}^*$ is the least set $\mathcal{S}(T)$ satisfying

$$\mathcal{S}(T) = \mathcal{A}(T) \cup \{kw \mid w \in \mathcal{S}(T),\ k \in \mathcal{K} \cap \mathcal{S}(T)\}. \tag{4}$$

The next lemmas follow immediately from Tarski's fixed-point theorem.

Lemma 2. *The analysis of a set of messages $T \subseteq \mathcal{K}^*$ is the union of the family of sets $\mathcal{A}_i(S)$ defined by*

$$\begin{aligned} \mathcal{A}_0(T) &= T \\ \mathcal{A}_{i+1}(T) &= \mathcal{A}_i(T) \cup \{w \mid kw \in \mathcal{A}_i(T),\ k \in \mathcal{K} \cap \mathcal{A}_i(T)\} \end{aligned}$$

Lemma 3. *The synthesis of a set of messages $T \subseteq \mathcal{K}^*$ is the union of the family of sets $\mathcal{S}_i(T)$ defined by*

$$\begin{aligned} \mathcal{S}_0(T) &= \mathcal{A}(T) \\ \mathcal{S}_{i+1}(T) &= \mathcal{S}_i(T) \cup \{kw \mid w \in \mathcal{S}_i(T),\ k \in \mathcal{K} \cap \mathcal{S}_i(T)\} \end{aligned}$$

The set of *compromised keys* K_c for a given set $T \subseteq \mathcal{K}^*$ of messages is defined by $K_c \stackrel{\text{def}}{=} \mathcal{K} \cap \mathcal{S}(T)$, which is easily seen to be equal to $\mathcal{K} \cap \mathcal{A}(T)$. (The compromised keys are immediately available for the intruder because they are either in his initial knowledge or can be discovered by the analysis.)

Remark 4. Let $T \subseteq \mathcal{K}^*$ be a given set of messages. The following observation is easy to verify: in order to compute the complete set K_c of compromised keys of size n, it is enough to find messages $m_1, \ldots, m_n \in T$ such that when we analyze them in a sequence, we discover exactly all compromised keys. Formally, we define

$$K_c^0 \stackrel{\text{def}}{=} \emptyset \quad \text{and} \quad K_c^i \stackrel{\text{def}}{=} K_c^{i-1} \cup \{k \in \mathcal{K} \mid m_i = wk,\ w \in (K_c^{i-1})^*\}$$

for all i, $1 \leq i \leq n$, and then $K_c = K_c^n$. □

Proposition 1. *It holds that $w \in \mathcal{S}(T)$ if and only if w can be written as $w = uw'$ for some $u \in K_c^*$ and there exists $u' \in K_c^*$ such that $u'w' \in T$.*

Proof. Notice that because of Lemma 2 $w \in \mathcal{A}(T)$ iff there is $u \in K_c^*$ such that $uw \in T$. The proposition then follows by an application of Lemma 3. □

We can now design an environment sensitive semantics for our calculus close in style to that of [4]. We define the reduction relation $\rightarrow$ by the following set of axioms (here $x \in P$ means that x is a summand in the defining equation of the process constant P).

$$
\begin{array}{rll}
(P \parallel C, T) \rightarrow (\overline{w\alpha}.P' \parallel C, T) & \text{if } (v\triangleright\,.\,\overline{w\triangleright}.P') \in P \text{ and } v\alpha \in \mathcal{S}(T) & \text{(A1)} \\
(P \parallel C, T) \rightarrow (P' \parallel C, T) & \text{if } (v.P') \in P \text{ and } v \in \mathcal{S}(T) & \text{(A2)} \\
(\overline{w}.P \parallel C, T) \rightarrow (P \parallel C, T \cup \{w\}) & & \text{(A3)} \\
(P \parallel C, T) \rightarrow (P' \parallel C, T \cup \{w\}) & \text{if } (\overline{w}.P') \in P & \text{(A4)}
\end{array}
$$

We show that this semantics can be internalized in our calculus within our existing semantics.

4.1 Intruder implementation

Let Pr be an initial configuration of a given protocol Δ with its (finite) set of *protocol keys* K_p. Without loss of generality we may assume that Δ is *input-guarded*, meaning that it contains no input prefixes of the form $\triangleright$ (i.e., every input prefix is guarded by at least one encryption key; this is easily guaranteed by replacing every summand of the form $\triangleright.\,\overline{w\triangleright}.P_i$ with $\sum_{k\in K_p} k\triangleright\,.\,\overline{wk\triangleright}.P_i$, which can be easily seen to generate the same labelled transition system).

Given a protocol configuration Pr, we shall define the intruder as an extra parallel component in the configuration (and hence using the same syntax). The intruder will remember a set $K_c \subseteq K_p$ of the compromised keys (as defined above) plus he will use an extra set of keys $\{\ell_P, \ell_B, \ell_T\}$ of so called *locking keys* and another key $\#$ called a *separation key*.

We implement the intruder as an abstract set of operations that can modify two main data structures, a *pool* and a *buffer*. Note that this is a conceptual abstraction: in fact the set of all data is represented as a single message containing the total sequence of keys of messages that have been encountered. For this the structure pool will store all available messages separated by the key $\#$, and the buffer structure can be either empty or it can store a single message in order to enable its analysis and synthesis. Moreover (for technical reasons) all messages placed into the pool will be in the reversed order of encryption keys.

All communication between the intruder and the protocol Pr will pass through the buffer. The available operations are in Table 1. Note that all the corresponding process constants are parameterized by the set K_c of currently discovered compromised keys. The intuition is that *listen* can transfer any message offered by the protocol Pr and store it to the buffer. Similarly *output* can offer the content of the buffer for communication with the original protocol Pr. *Deposit* can transfer the message stored in buffer into the pool (the extra key $\#$ is used to separate the messages stored in the pool and the transfered message is stored into P in the reversed order). *Retrieve* can nondeterministically select a single message from the pool and copy it into the buffer. *Analyze* can provide an analysis of the current message in the buffer according to the set of compromised keys K_c (during this a new compromised key can be discovered and the set K_c is

Operation	Direction	Process constant
Listen	Protocol → Buffer	L^{K_c}
Output	Buffer → Protocol	O^{K_c}
Deposit	Buffer → Pool	D^{K_c}
Retrieve	Pool → Buffer	R^{K_c}
Analyze	Buffer → Buffer	A^{K_c}
Synthesize	Buffer → Buffer	S^{K_c}
Switch	no data operation	X^{K_c}

Table 1. Abstract operations of the intruder

updated accordingly). *Synthesize* can add an arbitrarily long sequence of compromised keys onto the message currently stored in the buffer. Finally, the *switch* process constant X^{K_c} allows for a nondeterministic selection of any one of the previously mentioned operations. In any configuration of the intruder, exactly one of the process constants from Table 1 is present as a parallel component.

All internal communication within the intruder uses one of the locking keys mentioned above as the outermost key in the communicated messages. Together with our assumption that the original protocol is input-guarded by the keys K_p, this guarantees that none of those messages are available for a communication with the protocol as they have to be unlocked first. Also observe that because the process constants from Table 1 are parameterized by $K_c \subseteq K_p$, we have exponentially many of them with respect to the size of the set K_p. This is for technical convenience only, and it is discussed in more detail in Conclusion.

Let us now define all the components of the intruder.

4.2 Buffer implementation

All operations of the intruder use their own private *buffer*. A buffer B_ℓ is a process which can temporarily store any message encrypted with the locking key ℓ.

$$B_\ell \stackrel{\text{def}}{=} \ell\triangleright\,.\,\overline{\ell\triangleright}.B_\ell \tag{5}$$

The intruder uses three such buffers. We assign them the names P (pool), B (buffer) and B_T (temporary buffer) as follows.

$$P = B_{\ell_P} \qquad B = B_{\ell_B} \qquad B_T = B_{\ell_T}$$

4.3 Initial configuration

The *initial intruder* with a given set of compromised keys K_c is described by the following configuration.

$$I_{init} = \overline{\ell_P\#}.P \parallel \overline{\ell_B}.B \parallel \overline{\ell_T}.B_T \parallel X^{K_c} \tag{6}$$

The prefix $\overline{\ell_P \#}$ ensures that the empty message ϵ is available on the pool initially, and this will make it possible to output any compromised key k by using the synthesis operation on ϵ to obtain $k\epsilon = k$. The other two buffers are simply initialized to empty by transmitting only the corresponding locking key.

An *intruder configuration* is any configuration I such that $I_{init} \rightarrow^* I$. The behaviour of the intruder will be described in such a way that any intruder configuration will always be of the form

$$C_P \parallel C_B \parallel C_{B_T} \parallel C_X$$

where C_P is either P or $\overline{\ell_P w}.P$ for some $w \in (K_p \cup \{\#\})^*$, C_B is either B or $\overline{\ell_B w}.B$ for some $w \in K_p^*$, C_{B_T} is either B_T or $\overline{\ell_T w}.B_T$ for some $w \in (K_p \cup \{\#\})^*$, and C_X is either Y or $\overline{w}.Y$ where $Y \in \{X^{K_c}, L^{K_c}, O^{K_c}, D^{K_c}, R^{K_c}, A^{K_c}, S^{K_c}\}$ and $w \in \mathcal{K}^*$.

As the original protocol is assumed to be input-guarded, in a configuration like $Pr \parallel I$ (where Pr is a protocol configuration and I is an intruder configuration), we will make sure that only a process constant Y in the C_X part of I can manipulate the available buffers in parts C_P, C_B and C_{B_T}.

4.4 Switching between operations

The intruder is always capable of choosing nondeterministically between any of the six operations in Table 1. In this way, the intruder can manipulate messages by any means necessary. This is expressed by the central component of the intruder, the switch process constant X^{K_c}, which is parameterized by the set of currently compromised keys K_c.

$$\begin{aligned} X^{K_c} \stackrel{\text{def}}{=} \quad & \ell_B.L^{K_c} && (7)\\ & + \ell_P \triangleright .\overline{\ell_P \triangleright}.O^{K_c} && (8)\\ & + \ell_P \triangleright .\overline{\ell_P \triangleright}.D^{K_c} && (9)\\ & + \ell_B.R^{K_c} && (10)\\ & + \ell_P \triangleright .\overline{\ell_P \triangleright}.A^{K_c} && (11)\\ & + \ell_P \triangleright .\overline{\ell_P \triangleright}.S^{K_c} && (12) \end{aligned}$$

As X^{K_c} is always in parallel with a pool of the form $\overline{\ell_P w}.P$, the following communication is possible $\overline{\ell_P w}.P \parallel X^{K_c} \stackrel{\ell_P w}{\longrightarrow} P \parallel \overline{\ell_P w}.Y \stackrel{\ell_P w}{\longrightarrow} \overline{\ell_P w}.P \parallel Y$ for any $Y \in \{O^{K_c}, D^{K_c}, A^{K_c}, S^{K_c}\}$. This corresponds to giving a control to the appropriate process constant Y (the use of pool to make the switch is here only for technical reasons and the pool content is untouched). The process constants L^{K_c} and R^{K_c} use a different switching method which guarantees that the state change is possible only if the buffer B is empty (i.e., it outputs only the key ℓ_B).

4.5 Listen to the protocol

As the intruder runs in parallel with the protocol configuration Pr, it may collect any message transmitted by Pr. It then encrypts the message by the locking key ℓ_B and stores the message into the buffer B. The control is returned to X^{K_c} afterwards.

$$L^{K_c} \stackrel{\text{def}}{=} \sum_{k \in K_p} k\triangleright \,.\, \overline{\ell_B k\triangleright}.X^{K_c} \tag{13}$$

Formally, if $m \in K_p^*$ and Pr can transmit m and become Pr', we can perform the following communication.

$$Pr \parallel B \parallel L^{K_c} \stackrel{m}{\longrightarrow} Pr' \parallel B \parallel \overline{\ell_B m}.X^{K_c} \stackrel{\ell_B m}{\longrightarrow} Pr' \parallel \overline{\ell_B m}.B \parallel X^{K_c}$$

4.6 Output a message to the protocol

The intruder may output any message stored in the buffer B by removing its locking key ℓ_B, thereby making the message available to the protocol.

$$O^{K_c} \stackrel{\text{def}}{=} \ell_B\triangleright \,.\, \overline{\triangleright}.{O'}^{K_c} \tag{14}$$

$${O'}^{K_c} \stackrel{\text{def}}{=} \overline{l_B}.X^{K_c} \tag{15}$$

Hence if Pr can receive a message $m \in K_p^*$ and become Pr' then we get the following communication.

$$Pr \parallel \overline{\ell_B m}.B \parallel O^{K_c} \stackrel{\ell_B m}{\longrightarrow} Pr \parallel B \parallel \overline{m}.{O'}^{K_c} \stackrel{m}{\longrightarrow}$$
$$Pr' \parallel B \parallel {O'}^{K_c} \stackrel{\ell_B}{\longrightarrow} Pr' \parallel \overline{\ell_B}.B \parallel X^{K_c}$$

4.7 Analysis and synthesis

Assume that the buffer B contains some message of the form $\overline{\ell_B w}.B$ where $w \in {K_p}^*$. Computing an element of the analysis amounts to step-by-step decryption with any of the compromised keys from K_c.

$$A^{K_c} \stackrel{\text{def}}{=} \sum_{k \in K_c} \ell_B k\triangleright \,.\, \overline{\ell_B\triangleright}.A^{K_c} + \sum_{k \in K_p} \ell_B k.{A_1}_k^{K_c} + \ell_P\triangleright \,.\, \overline{\ell_P\triangleright}.X^{K_c} \tag{16}$$

$${A_1}_k^{K_c} \stackrel{\text{def}}{=} \overline{\ell_B}.X^{K_c \cup \{k\}} \quad \text{for all } k \in K_p \tag{17}$$

The second summand in equation (16) corresponds to the discovery of a plaintext key, which is promptly added to the set of compromised keys by equation (17), the buffer B is initialized to $\overline{\ell_B}.B$ and the control is given to $X^{K_c \cup \{k\}}$. The third summand in (16) allows the intruder to end the analysis phase and return the control to the switch X^{K_c}.

Computing an element of the synthesis simply enables to add (step-by-step) as many compromised keys as the intruder wants to the message stored in the buffer B.

$$S^{K_c} \stackrel{\text{def}}{=} \sum_{k \in K_c} \ell_B \triangleright \,.\, \overline{\ell_B k \triangleright}.S^{K_c} + \ell_P \triangleright \,.\, \overline{\ell_P \triangleright}.X^{K_c} \tag{18}$$

4.8 Deposit

This operation deposits a message from the buffer B into the pool P by moving it key by key.

$$D^{K_c} \stackrel{\text{def}}{=} \sum_{k \in K_p} \ell_B k \triangleright \,.\, \overline{\ell_B \triangleright}.D_k^{K_c} + \ell_B.D'^{K_c}_{\#} \tag{19}$$

$$D_k^{K_c} \stackrel{\text{def}}{=} \ell_P \triangleright \,.\, \overline{\ell_P k \triangleright}.D^{K_c} \quad \text{for all } k \in K_p \tag{20}$$

$$D'^{K_c}_{\#} \stackrel{\text{def}}{=} \overline{\ell_B}.D_{\#}^{K_c} \tag{21}$$

$$D_{\#}^{K_c} \stackrel{\text{def}}{=} \ell_P \triangleright \,.\, \overline{\ell_P \# \triangleright}.X^{K_c} \tag{22}$$

The equations (19) and (20) perform the key transfer from B to P. One step of such a transfer looks as follows (here $k \in K_p$ and w_1 and w_2 are arbitrary messages).

$$\overline{\ell_P w_1}.P \parallel \overline{\ell_B k w_2}.B \parallel D^{K_c} \xrightarrow{\ell_B k w_2} \overline{\ell_P w_1}.P \parallel B \parallel \overline{\ell_B w_2}.D_k^{K_c} \xrightarrow{\ell_B w_2}$$

$$\overline{\ell_P w_1}.P \parallel \overline{\ell_B w_2}.B \parallel D_k^{K_c} \xrightarrow{\ell_P w_1} P \parallel \overline{\ell_B w_2}.B \parallel \overline{\ell_P k w_1}.D^{K_c} \xrightarrow{\ell_P k w_1}$$

$$\overline{\ell_P k w_1}.P \parallel \overline{\ell_B w_2}.B \parallel D^{K_c}$$

Recall that as indicated before, the messages are stored in reverse order in P. When the bottom of the buffer B is reached, the second summand in equation (19) can be applied. First the key ℓ_B is returned to the buffer B by equation (21) and finally the control is given to the process constant X^{K_c} while adding the separation key # into the pool by equation (22).

4.9 Retrieve

To retrieve a message from the pool (the most complicated operation of the construction), the retrieval process scans through the message separation markers # until it nondeterministically decides to copy a message to the buffer (here it is also the only place where the temporary buffer is used).

$$R^{K_c} \stackrel{\text{def}}{=} \overline{\ell_B}.{R_1}^{K_c} \tag{23}$$

$${R_1}^{K_c} \stackrel{\text{def}}{=} \sum_{k \in K_p \cup \{\#\}} \ell_P k \triangleright \,.\, \overline{\ell_P \triangleright}.R_k^{K_c} + \ell_P \# \triangleright \,.\, \overline{\ell_P \triangleright}.S_\#^{K_c} + \ell_P.{T_2'}^{K_c} \tag{24}$$

$$R_k^{K_c} \stackrel{\text{def}}{=} \ell_T \triangleright \,.\, \overline{\ell_T k \triangleright}.{R_1}^{K_c} \quad \text{for all } k \in K_p \cup \{\#\} \tag{25}$$

$$S_\#^{K_c} \stackrel{\text{def}}{=} \ell_T \triangleright \,.\, \overline{\ell_T \# \triangleright}.T^{K_c} \tag{26}$$

$$T^{K_c} \stackrel{\text{def}}{=} \sum_{k \in K_p} \ell_P k \triangleright \,.\, \overline{\ell_P \triangleright}.T_k^{K_c} + \ell_P \# \triangleright \,.\, \overline{\ell_P \# \triangleright}.{T_2}^{K_c} + \ell_P.{T_2'}^{K_c} \tag{27}$$

$$T_k^{K_c} \stackrel{\text{def}}{=} \ell_T \triangleright \,.\, \overline{\ell_T k \triangleright}.{T_1}_k^{K_c} \quad \text{for all } k \in K_p \tag{28}$$

$${T_1}_k^{K_c} \stackrel{\text{def}}{=} \ell_B \triangleright \,.\, \overline{\ell_B k \triangleright}.T^{K_c} \quad \text{for all } k \in K_p \tag{29}$$

$${T_2'}^{K_c} \stackrel{\text{def}}{=} \overline{\ell_P}.{T_2}^{K_c} \tag{30}$$

$${T_2}^{K_c} \stackrel{\text{def}}{=} \sum_{k \in K_p \cup \{\#\}} \ell_T k \triangleright \,.\, \overline{\ell_T \triangleright}.{T_3}_k^{K_c} + \ell_T.{T_4}^{K_c} \tag{31}$$

$${T_3}_k^{K_c} \stackrel{\text{def}}{=} \ell_P \triangleright \,.\, \overline{\ell_P k \triangleright}.{T_2}^{K_c} \quad \text{for all } k \in K_p \cup \{\#\} \tag{32}$$

$${T_4}^{K_c} \stackrel{\text{def}}{=} \overline{\ell_T}.X^{K_c} \tag{33}$$

First the buffer B is initialized with the key ℓ_B by equation (23). Then by equations (24) and (25) all keys from K_p can be moved one by one to the temporary buffer B_T. In case that the #-key is discovered during the process, the intruder can either continue as before by placing the key into the temporary buffer (using the first summand in (24)), or he can decide that # is the start of the selected message to be copied into the buffer B and use the second summand of (24).

In the latter case equation (26) places the # key into the temporary buffer B_T and evokes the process constant T^{K_c} which copies the next piece of the message from the pool both into the temporary buffer B_T and also into the buffer B. This is done using the equations (28) and (29) and the control returns to T^{K_c}. Whenever the first separation key # appears as the outermost key, the second summand in (27) applies and the intruder continues from ${T_2}^{K_c}$ in equation (31). The same happens when the end of the pool is reached (third summand of equation (27)). In this case the key ℓ_P is first return into the pool by the process constant ${T_2'}^{K_c}$ and then the process continues by equation (31) as before.

The remaining equations (31) – (33) simply return the content of the temporary buffer B_T back to the pool and give the control to X^{K_c} (making sure that the temporary buffer is reseted to $\overline{\ell_T}.B_T$).

4.10 Correctness of the encoding

We now show that the intruder faithfully describes the environment sensitive semantics defined in the beginning of Section 4 by axioms (A1) – (A4).

To do this, we need to define the knowledge of the intruder. We take this to be the totality of the set of messages that he can output to the environment.

Let $E \stackrel{\text{def}}{=} \sum_{k \in K_p} k\rhd \,.\, \overline{k\rhd}.E$ be an observer (environment) which allows us to observe all messages encrypted by keys from K_p. Let $\Longrightarrow$ be an (unlabelled) reflexive and transitive closure of transition relations labelled by actions that are not available to the environment (internal communication of the intruder), i.e.,

$$\Longrightarrow \stackrel{\text{def}}{=} \Big(\bigcup_{m \in (\mathcal{K} \setminus K_p) \cdot \mathcal{K}^*} \stackrel{m}{\longrightarrow} \Big)^*.$$

The set of *known messages* of an intruder configuration I is defined by

$$km(I) \stackrel{\text{def}}{=} \{w \in {K_p}^* \mid E \parallel I \Longrightarrow \circ \stackrel{w}{\longrightarrow}\}.$$

We say that I is a *proper* intruder configuration whenever I does not contain any parallel component of the form L^{K_c}, O^{K_c} or $\overline{m}.{O'}^{K_c}$. In other words it is not directly committed to communicate with the environment.

We now show an operational correspondence result which states that the intruder, as described above, accurately describes the behaviour of the environment sensitive semantics expressed by axioms (A1) – (A4). The following lemma shows that the intruder can synthesize all possible messages.

Lemma 4. *For any proper intruder configuration I it holds that* $\mathrm{km}(I) = \mathcal{S}(\mathrm{km}(I))$.

Proof. The inclusion $km(I) \subseteq \mathcal{S}(km(I))$ follows from the definition of synthesis. Let $w \in \mathcal{S}(km(I))$. We will show that $w \in km(I)$. Because of Remark 4 we know that the set of compromised keys K_c (where $n = |K_c|$) can be computed by analyzing of certain messages $m_1, \ldots, m_n \in km(I)$. As these messages belong to $km(I)$ they can consequently be placed (in the given order) into the buffer B and analyzed. This will update the current set of compromised keys remembered by the intruder to the complete set K_c. Because of Proposition 1 we know that w can be constructed by first placing a selected message into the buffer and by removing all the compromised keys (by A^{K_c}) and then by adding the compromised keys in order to form w (by S^{K_c}). The message w can then be communicated to the environment E by O^{K_c} and hence $w \in km(I)$. □

The first theorem states that the intruder can mimic the behaviour of the environment-sensitive semantics.

Theorem 3 (Completeness). *Let C be a protocol configuration, let $T \subseteq \mathcal{K}^*$, and let I be a proper intruder configuration such that $\mathcal{S}(T) = \mathrm{km}(I)$. If*

$$(C, T) \rightarrow (C', T')$$

then there exists $w \in K_p^*$ *and a transition sequence*

$$(C \parallel I) \Longrightarrow \circ \xrightarrow{w} (C' \parallel I')$$

such that $\mathcal{S}(T') = \mathrm{km}(I')$ *and* I' *is a proper intruder configuration.*

Proof. The intruder's strategy is clear. He listens to every message communicated by the protocol and stores all such messages into the pool in order to be able to synthesize the appropriate messages and communicate them back to the protocol at some later stage.

First observe that there are four possible axioms (A1) – (A4) to perform $(C, T) \to (C', T')$. In case of the axioms (A1) and (A2) we know that the protocol configuration C receives a message w from $\mathcal{S}(T)$ and becomes C', and that $T' = T$. Because $w \in \mathcal{S}(T) = km(I)$ we know that the intruder I can perform $I \Longrightarrow I''$ such that $E \parallel I'' \xrightarrow{w} E' \parallel I'$ and hence also $(C \parallel I) \Longrightarrow \circ \xrightarrow{w} (C' \parallel I')$. Moreover, $km(I') = km(I)$ as required because every message that appears in the buffer B can be also copied to the pool P for a use later on. It is also easy to see that I' is proper.

In case of the axioms (A3) and (A4) the protocol configuration C emits a message w and becomes C', and $T' = T \cup \{w\}$. In this case the intruder deposits the content of the buffer into the pool (in case that it was not empty) and then receives the message w from C. This means that $(C \parallel I) \Longrightarrow \circ \xrightarrow{w} (C' \parallel I')$ and moreover $km(I')$ now contains the message w (obviously I' is proper). This implies that $\mathcal{S}(T') = \mathcal{S}(T \cup \{w\}) = km(I')$ because of Lemma 4.

□

Conversely, the intruder cannot gain any extra knowledge than the one he can construct by analysis and synthesis of the available messages.

Theorem 4 (Soundness). *Let* C *be a protocol configuration, let* $T \subseteq \mathcal{K}^*$*, and let* I *be an intruder configuration such that* $\mathrm{km}(I) \subseteq \mathcal{S}(T)$*. If for some* $w \in K_p^*$

$$(C \parallel I) \Longrightarrow \circ \xrightarrow{w} (C' \parallel I')$$

then there exists a transition

$$(C, T) \to (C', T') \qquad \textit{or} \qquad (C, T) \to \circ \to (C', T')$$

such that $\mathrm{km}(I') \subseteq \mathcal{S}(T')$*.*

Proof. There are two cases to distinguish. In the first case the intruder either listens to the message w sent by C, or he outputs w for a communication with the protocol C. This means that there is a corresponding transition also in $(C, T) \to (C', T')$ and it is easy to verify that $km(I') \subseteq \mathcal{S}(T')$. In the second case the intruder does not interfere with the communication in C (hence $km(I') = km(I)$) and such a communication step is simulated by two steps $(C, T) \to \circ \to (C', T')$ (in the environment sensitive semantics all messages must pass through T). By the fact that $\mathcal{S}(T) \subseteq \mathcal{S}(T')$, we can immediately see that $km(I') \subseteq \mathcal{S}(T')$. □

5 Replicative ping-pong protocols

In this section we shall define a replicative variant of our calculus for ping-pong protocols. We will then show that this formalism is not Turing powerful because the reachability problem becomes decidable. This is to be put in contrast with several results where replicative calculi are known to be capable of simulating recursive ones (see e.g. [17] for the case of pi-calculus which implies also the same result for spi-calculus, and [13, 18] for other examples). On the other hand, a similar discrimination result as ours between recursion and replication was recently proved for CCS [7].

Let us now define *replicative ping-pong protocols*. Let $\mathcal{K}$ be the set of encryption keys as before. The set $\mathcal{C}onf$ of protocol configurations is given by the following abstract syntax

$$C \ ::= \ \mathbf{0} \ \mid \ v\triangleright \,.\, \overline{w\triangleright} \ \mid \ v \ \mid \ \overline{w} \ \mid !(v\triangleright \,.\, \overline{w\triangleright}) \ \mid !(v) \ \mid !(\overline{w}) \ \mid \ C \parallel C$$

where $\mathbf{0}$ is the symbol for the empty configuration, v and w range over $\mathcal{K}^*$, and ! is the bang operator (replication). As before, we shall introduce structural congruence $\equiv$, which is the smallest congruence over $\mathcal{C}onf$ such that

- $(\mathcal{C}onf, \parallel, \mathbf{0})$ is a commutative monoid
- $\epsilon \parallel C \equiv C \equiv \ \overline{\epsilon} \parallel C$
- $!(\epsilon) \equiv \mathbf{0} \equiv !(\overline{\epsilon})$
- $!(C) \equiv C \parallel !(C)$.

A labelled transition system determined by a configuration (where states are configurations modulo $\equiv$ and labels are non-empty messages as before) is defined by the following SOS rules (recall the replicative axiom $!(C) \equiv C \parallel !(C)$ and the fact that '$\parallel$' is commutative).

$$\frac{m \in \mathcal{K}^+}{\overline{m} \parallel m \parallel C \ \stackrel{m}{\longrightarrow} \ C} \qquad \frac{m \in \mathcal{K}^+ \qquad m = v\alpha}{\overline{m} \parallel v\triangleright \,.\, \overline{w\triangleright} \parallel C \ \stackrel{m}{\longrightarrow} \ \overline{w\alpha} \parallel C}$$

Example 3. An initial configuration $C \stackrel{\text{def}}{=} !(\overline{k}) \ \parallel \ !(k\triangleright \,.\, \overline{kk\triangleright})$ generates a labelled transition system in Figure 2 with infinitely many reachable configurations (only a finite fragment is depicted). Observe that (unlike recursive protocols) the number of parallel components can grow arbitrarily (e.g. the left-most branch in the picture). The reason is that we allow prefixes of the form $!(\overline{w})$ which can unconditionally output new messages and that we have replicative agents accepting these messages.

In case that the number of output prefixes for all reachable configurations is bounded by some number n, the parallel components of the form $!(v)$, $!(\overline{w})$ and $!(v \triangleright \,.\, \overline{w\triangleright})$ can be replaced by n parallel occurrences of fresh process constants $P_{!(v)}$, $P_{!(\overline{w})}$ and $P_{!(v\triangleright .\, \overline{w\triangleright})}$ respectively such that $P_{!(v)} \stackrel{\text{def}}{=} v.P_{!(v)}$, $P_{!(\overline{w})} \stackrel{\text{def}}{=} \overline{w}.P_{!(\overline{w})}$ and $P_{!(v\triangleright .\, \overline{w\triangleright})} \stackrel{\text{def}}{=} v\triangleright \,.\, \overline{w\triangleright}.P_{!(v\triangleright .\, \overline{w\triangleright})}$, and hence it can be simulated by recursive protocols. □

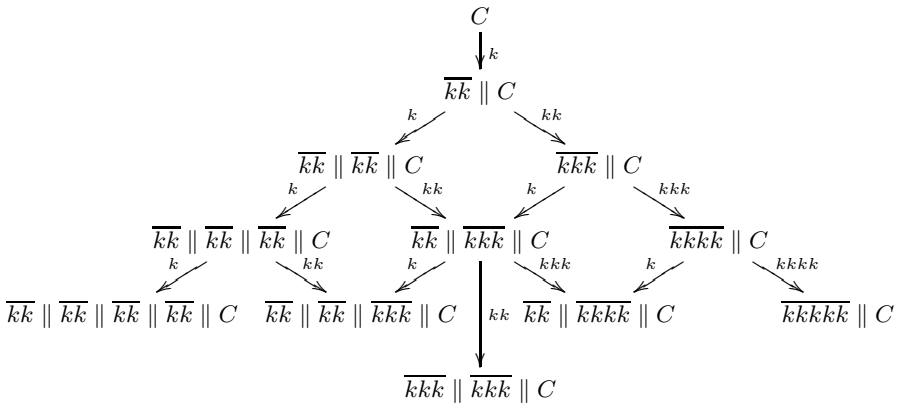

Fig. 2. Initial fragment of a labelled transition system for $C \stackrel{\text{def}}{=} !(\overline{k}) \parallel !(k\triangleright\,.\,\overline{kk\triangleright})$

We shall now show that the reachability problem for general replicative ping-pong protocols is decidable. We reduce our problem to reachability of weak process rewrite systems (wPRS) which was very recently proven to be decidable [15].

Following the work of Mayr [16], let $\mathcal{C}onst$ be a set of *process constants*. The set $\mathcal{E}$ of *process expressions* over $\mathcal{C}onst$ is defined by

$$E ::= \epsilon \;\mid\; X \;\mid\; E.E \;\mid\; E \parallel E$$

where ϵ is the symbol for empty expression, X ranges over $\mathcal{C}onst$, '.' is the operator of sequential composition and '$\parallel$' is the operator of parallel composition. We identify processes up to structural congruence, which is the least congruence such that '.' is associative, '$\parallel$' is associative and commutative and ϵ is a unit for '.' and '$\parallel$'.

Let Q be a finite set of *control-states* and $\mathcal{A}ct$ a set of *actions*. A *state-extended process rewrite system* (sePRS) is a finite set Δ of rewrite rules of the form $pE \stackrel{a}{\longrightarrow} qF$ where $p, q \in Q$, $a \in \mathcal{A}ct$, and $E, F \in \mathcal{E}$ such that $E \neq \epsilon$.

A given sePRS Δ generates a labelled transition system $T(\Delta)$ where states are process expressions over $\mathcal{C}onst$ modulo the structural congruence, the set of actions is $\mathcal{A}ct$ and the transition relation is given by the following SOS rules (recall that '$\parallel$' is commutative).

$$\frac{(pE \stackrel{a}{\longrightarrow} qF) \in \Delta}{pE \stackrel{a}{\longrightarrow} qF} \qquad \frac{pE \stackrel{a}{\longrightarrow} qE'}{p(E.F) \stackrel{a}{\longrightarrow} q(E'.F)} \qquad \frac{pE \stackrel{a}{\longrightarrow} qE'}{p(E \parallel F) \stackrel{a}{\longrightarrow} q(E' \parallel F)}$$

A sePRS Δ is called a *weak process rewrite system* (wPRS) iff there is a partial ordering $\leq$ on Q such that all rewrite rules $pE \stackrel{a}{\longrightarrow} qF$ from Δ satisfy that $q \leq p$.

Theorem 5 ([15]). *The reachability problem for wPRS is decidable.*

Let us now consider an arbitrary configuration C in the calculus of replicative ping-pong protocols. We shall construct a wPRS system Δ which preserves the reachability property.

The configuration C can be naturally written as $C \equiv A \parallel B \parallel O$ where A contains all parallel components of the form $!(v\triangleright\,.\,\overline{w\triangleright})$, $!(v)$ and $!(\overline{w})$; B contains all parallel components of the form $v\triangleright\,.\,\overline{w\triangleright}$ and v; and O contains all output prefixes of the form $\overline{w}$. Here we assume that the rules of the structural congruence $\equiv$ are applied as long as possible in order to minimize the size of the configuration C (such assumptions are implicit also later on). Observe now that any configuration $C' \equiv A \parallel B' \parallel O'$ reachable from C contains exactly the same part A and every parallel component from B' is also in B.

The intuition of the reduction is that A does not have to be remembered as all parallel components from A are always available, B will be stored in control-states of the wPRS (note that there are only finitely many different components in B' for all reachable configurations of the form $A \parallel B' \parallel O'$) and the parallel components from O will be stored as a parallel composition of stacks in the wPRS system.

Let $C \equiv A \parallel B \parallel O$ be the initial configuration. Formally, the wPRS rules Δ where $\mathcal{C}onst \stackrel{\text{def}}{=} \mathcal{K} \cup \{Z, X\}$ (Z is a special symbol for the bottom of a stack, X is a process constant for creating more parallel components) and where $Q \stackrel{\text{def}}{=} \{p^{B'} \mid \exists B'' \text{ s.t. } B \equiv B' \parallel B''\}$ are defined as follows.

1.	$p^{B'}X \longrightarrow p^{B'}(X \parallel w.Z)$	$B' \subseteq B$ and $!(\overline{w}) \in A$
2.	$p^{B'}w.Z \longrightarrow p^{B'}$	$B' \subseteq B$ and $!(\overline{w}) \in A$
3.	$p^{B'}Z \longrightarrow p^{B'}$	$B' \subseteq B$
4.	$p^{B'}v.Z \longrightarrow p^{B'}$	$B' \subseteq B$ and $!(v) \in A$
5.	$p^{B'}v \longrightarrow p^{B'}w$	$B' \subseteq B$ and $!(v\triangleright\,.\,\overline{w\triangleright}) \in A$
6.	$p^{B'}v.Z \longrightarrow p^{B''}$	$B' \subseteq B$ and $B' \equiv B'' \parallel v$
7.	$p^{B'}v \longrightarrow p^{B''}w$	$B' \subseteq B$ and $B' \equiv B'' \parallel v\triangleright\,.\,\overline{w\triangleright}$

In the definitions above, whenever we have $w \in \mathcal{K}^*$, we use the word w also in the wPRS rules in the meaning that it represents a sequential composition of process constants contained in w, i.e., if $w = a_1a_2\cdots a_n$ then w in the wPRS rules stands for the sequential composition $a_1.a_2.\ldots.a_n$. Moreover $B' \subseteq B$ means that there is some B'' such that $B \equiv B' \parallel B''$ and $x \in A$ means that the expression x is a parallel component in A. All actions are omitted as they are irrelevant for the reachability question.

Rules 1. – 3. correspond to the structural congruence $\equiv$. As $!(\overline{w}) \equiv \overline{w} \parallel !(\overline{w})$ rule 1. enables us to create a new parallel component $\overline{w}$ whenever $!(\overline{w}) \in A$

and by rule 2. such a component can always be deleted. Rule 3. corresponds to the fact that $\epsilon \parallel C \equiv C$. (Recall that we assume that C does not contain any component of the form $!(\epsilon)$ or $!(\overline{\epsilon})$.)

Rules 4. – 7. are computational rules: in rules 4. and 5. we allow the reception of messages by the components in A and the control-state does not change in this case; in rules 6. and 7. we do the same for components in B' (the current remaining part of B) and whenever such a component is used, we remove it from B' by changing the control-state to $p^{B''}$.

Let us assume the initial configuration $C \equiv A \parallel B \parallel O$ as above such that $O \equiv \overline{w_1} \parallel \overline{w_2} \parallel \cdots \parallel \overline{w_n}$. The initial configuration of the wPRS system Δ is then

$$p^B(X \parallel w_1.Z \parallel w_2.Z \parallel \cdots \parallel w_n.Z).$$

It is easy to see that every rewriting step in Δ corresponds either to a single computational step in the replicative ping-pong protocol or to an application of some congruence rule. On the other hand, any communication in the protocol can be directly simulated in Δ.

Hence we can make the following observation (assuming that $O' \equiv \overline{w'_1} \parallel \overline{w'_2} \parallel \cdots \parallel \overline{w'_{n'}}$).

Lemma 5. *It holds that*

$$A \parallel B \parallel O \longrightarrow^* A \parallel B' \parallel O'$$

if and only if

$$p^B(X \parallel w_1.Z \parallel w_2.Z \parallel \cdots \parallel w_n.Z) \longrightarrow^* p^{B'}(X \parallel w'_1.Z \parallel w'_2.Z \parallel \cdots \parallel w'_{n'}.Z).$$

Theorem 6. *The reachability problem for replicative ping-pong protocols is decidable.*

Proof. By Lemma 5 we reduced the problem to the reachability question for wPRS (observe that $p^{B'} \geq p^{B''}$ iff $B'' \subseteq B'$ is the natural ordering on control-states of the wPRS demonstrating that the control-state unit has a monotone behaviour). The decidability result then follows from Theorem 5. □

6 Conclusion

We have seen that ping-pong protocols extended with recursive definitions have full Turing power. This is the case even in the absence of nondeterministic choice operator '+'. A result like this implies that any reasonable property for all richer calculi cannot be automatically verified.

We also presented an explicit description of the active intruder in the syntax of recursive ping-pong protocols. The presented encoding gives an exponential blow-up in the size of the protocol. However, it is in fact not necessary and for a given protocol an active intruder of polynomial size can be constructed (it is enough to store the set of compromised keys in an extra buffer). We have decided

to present the construction which is exponentially larger for clarity reasons. Nevertheless the polynomial modification of the analysis and synthesis part using the extra buffer can be easily implemented – it only becomes technically more tedious.

Finally, we showed that reachability analysis for a replicative variant of the protocol becomes feasible. Our proof uses very recent results from process algebra [15] and can be compared to the work of Amadio, Lugiez and Vanackère [4] which establishes the decidability of reachability for a similar replicative protocol capable of ping-pong behaviour. Their approach uses a notion of a pool of messages explicitly modelled in the semantics and reduces the question to a decidable problem of reachability for prefix rewriting. In our approach we allow spontaneous generation of new messages which is not possible in their calculus. Moreover, we can distinguish between replicated and once-only behaviours (unlike in [4] where all processes have to be replicated).

Last but not least we hope that our approach can be possibly extended to include other operations as the decidability result for replicative protocols uses only a limited power of wPRS (only a parallel composition of stacks). Hence there is a place for further extensions of the protocol syntax while preserving a decidable calculus (e.g. messages of the form $\overline{k_1(k_2\ op\ k_3)k_4}$ for some extra composition operation *op* on keys can be easily stored in wPRS as $k_1.(k_2 \parallel k_3).k_4$). Such a study is left for future research.

Other open problems include decidability of bisimilarity for replicative ping-pong protocols and the question to determine general conditions that guarantee equi-expressiveness of recursion and replication (see e.g. [17, 13, 18]).

References

1. M. Abadi and A.D. Gordon. A bisimulation method for cryptographic protocols. *Nordic Journal of Computing*, 5(4):267–303, 1998.
2. R.M. Amadio and W. Charatonik. On name generation and set-based analysis in the Dolev-Yao model. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *LNCS*, pages 499–514. Springer-Verlag, 2002.
3. R.M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *LNCS*, pages 380–394. Springer-Verlag, 2000.
4. R.M. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, October 2002.
5. M. Boreale. Symbolic trace analysis of cryptographic protocols. In *Proceedings of the 28th Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *LNCS*, pages 667–681. Springer, July 2001.
6. O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 9, pages 545–623. Elsevier Science, 2001.
7. N. Busi, M. Gabbrielli, and G. Zavattaro. Replication vs. recursive definitions in channel based calculi. In *Proceedings of the 30th International Colloquium on*

Automata, Languages, and Programming (ICALP'03), volume 2719 of *LNCS*, pages 133–144. Springer-Verlag, 2003.
8. D. Dolev, S. Even, and R.M. Karp. On the security of ping-pong protocols. *Information and Control*, 55(1–3):57–68, 1982.
9. D. Dolev and A.C. Yao. On the security of public key protocols. *Transactions on Information Theory*, IT-29(2):198–208, 1983.
10. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and E. Clarke, editors, *Proceedings of Workshop on Formal Methods and Security Protocols (FMSP'99)*, July 1999.
11. M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *14th IEEE Computer Security Foundations Workshop (CSFW '01)*, pages 160–173, Washington - Brussels - Tokyo, June 2001. IEEE.
12. R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP'00)*, volume 1853 of *LNCS*, pages 354–372. Springer-Verlag, 2000.
13. P. Giambiagi, G. Schneider, and F.D. Valencia. On the expressiveness of infinite behavior and name scoping in process calculi. In *Proceedings of the 7nd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume 2987 of *LNCS*, pages 226–240. Springer-Verlag, 2004.
14. H. Hüttel and J. Srba. Recursive ping-pong protocols. In *Proceedings of the 4th International Workshop on Issues in the Theory of Security (WITS'04)*, pages 129–140, 2004.
15. M. Křetínský, V. Řehák, and J. Strejček. Extended process rewrite systems: Expressiveness and reachability. In *Proceedings of the 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *LNCS*, pages 355–370. Springer-Verlag, 2004.
16. R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.
17. R. Milner. The polyadic pi-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.
18. M. Nielsen, C. Palamidessi, and F.D. Valencia. On the expressive power of temporal concurrent constraint programming languages. In *Proceedings of the 4th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 156–167. ACM Press, 2002.
19. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *TCS: Theoretical Computer Science*, 299, 2003.
20. R.J. van Glabbeek. The linear time - branching time spectrum I: The semantics of concrete, sequential processes. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier Science, 2001.

Recent BRICS Report Series Publications

RS-04-23 **Hans Hüttel and Jiří Srba.** ***Recursion vs. Replication in Simple Cryptographic Protocols*****. October 2004. 26 pp. To appear in Vojtas, editor,** ***31st Conference on Current Trends in Theory and Practice of Informatics*****, SOFSEM '05 Proceedings, LNCS, 2005.**

RS-04-22 **Gian Luca Cattani and Glynn Winskel.** ***Profunctors, Open Maps and Bisimulation*****. October 2004. 64 pp. To appear in** ***Mathematical Structures in Computer Science*****.**

RS-04-21 **Glynn Winskel and Francesco Zappa Nardelli.** ***New-HOPLA—A Higher-Order Process Language with Name Generation*****. October 2004. 38 pp.**

RS-04-20 **Mads Sig Ager.** ***From Natural Semantics to Abstract Machines*****. October 2004. 21 pp. Presented at the** ***International Symposium on Logic-based Program Synthesis and Transformation*****, LOPSTR 2004, Verona, Italy, August 26–28, 2004.**

RS-04-19 **Bolette Ammitzbøll Madsen and Peter Rossmanith.** ***Maximum Exact Satisfiability: NP-completeness Proofs and Exact Algorithms*****. October 2004. 20 pp.**

RS-04-18 **Bolette Ammitzbøll Madsen.** ***An Algorithm for Exact Satisfiability Analysed with the Number of Clauses as Parameter*****. September 2004. 4 pp.**

RS-04-17 **Mayer Goldberg.** ***Computing Logarithms Digit-by-Digit*****. September 2004. 6 pp.**

RS-04-16 **Karl Krukow and Andrew Twigg.** ***Distributed Approximation of Fixed-Points in Trust Structures*****. September 2004. 25 pp.**

RS-04-15 **Jesús Fernando Almansa.** ***Full Abstraction of the UC Framework in the Probabilistic Polynomial-time Calculus ppc*****. August 2004.**

RS-04-14 **Jesper Makholm Byskov.** ***Maker-Maker and Maker-Breaker Games are PSPACE-Complete*****. August 2004. 5 pp.**

RS-04-13 **Jens Groth and Gorm Salomonsen.** ***Strong Privacy Protection in Electronic Voting*****. July 2004. 12 pp. Preliminary abstract presented at Tjoa and Wagner, editors,** ***13th International Workshop on Database and Expert Systems Applications*****, DEXA '02 Proceedings, 2002, page 436.**