Bounded Memory Protocols and Progressing Collaborative Systems

Max Kanovich[1], Tajana Ban Kirigin[2], Vivek Nigam[3], and Andre Scedrov[4]

[1] Queen Mary, University of London, UK,
mik@dcs.qmul.ac.uk
[2] University of Rijeka, HR
bank@math.uniri.hr
[3] Federal University of Paraíba, João Pessoa, Brazil
vivek@ci.ufpb.br
[4] University of Pennsylvania, Philadelphia, USA
scedrov@math.upenn.edu

Abstract. It is well-known that the Dolev-Yao adversary is a powerful adversary. Besides acting as the network, intercepting, sending, and composing messages, he can remember as much information as he needs. That is, his memory is unbounded. We recently proposed a weaker Dolev-Yao like adversary, which also acts as the network, but whose memory is bounded. We showed that this Bounded Memory Dolev-Yao adversary, when given enough memory, can carry out many existing protocol anomalies. In particular, the known anomalies arise for *bounded memory protocols*, where there is only a bounded number of concurrent sessions and the honest participants of the protocol cannot generate an unbounded number of facts nor an unbounded number of nonces. This led us to the question of whether it is possible to infer an upper-bound on the memory required by the Dolev-Yao adversary to carry out an anomaly from the memory restrictions of the bounded protocol. This paper answers this question negatively. The second contribution of this paper is the formalization of Progressing Collaborative Systems that may create fresh values, such as nonces. We also prove the NP-completeness of the reachability problem for Progressing Collaborative Systems that may create fresh values.

1 Introduction

In the symbolic verification of protocol security, one considers a powerful adversary model now usually referred to as the Dolev-Yao adversary, which arose from positions taken by Needham and Schroeder [16] and a model presented by Dolev and Yao [6]. Not only can the Dolev-Yao adversary act as the network, intercepting, sending and composing messages, but he can also remember as much information as he needs. The goal in protocol verification is to demonstrate that such a powerful adversary cannot discover some secret information, when using some protocol(s). Clearly, if it is shown that such a powerful adversary cannot discover symbolically the secret, then weaker adversaries will also not be able to discover the secret.

In [10], we proposed a Bounded Memory Dolev-Yao adversary, which is very similar to the Dolev-Yao adversary. He also acts as the network, intercepting, sending and composing messages, but differently from the Dolev-Yao adversary, he can remember only a bounded number of facts at a given time. So, in order for him to learn some new information, such as a nonce, he might have to forget some information he previously learned. Clearly, our Bounded Memory Dolev-Yao adversary is weaker than the Dolev-Yao adversary, as the former's memory is bounded, while the latter's is not.

However, despite being weaker, we also demonstrated in [10] that many known anomalies can also be carried out by our Bounded Memory Dolev-Yao adversary. We also noticed that the protocols for which we could replay the anomaly with our bounded memory adversary were all *bounded memory protocols*. In bounded memory protocols, one considers that the memory of the system is bounded. That is, in concurrent runs the honest participants of the protocol also cannot generate an unbounded number of facts nor an unbounded number of nonces. This led us to the question of whether it is possible to infer an upper bound on the memory of the Dolev-Yao adversary with respect to the memory restrictions of bounded memory protocols, that is, of the memory used by the participants.

This paper answers this question negatively. That is, it is not possible to determine an upper bound on the memory of the Dolev-Yao adversary even if the memory of the protocol is bounded. We show this negative result by proposing a novel undecidability proof for the secrecy problem with the Dolev-Yao adversary. Our undecidability result strengthens the one given in [7], confirming the hardness of protocol verification. In particular, we show that the secrecy problem is very undecidable: the secrecy problem is undecidable *even for bounded memory protocols* and thus a bound on the memory of the Dolev-Yao adversary is not computable from a bound on the memory used by a protocol. This is accomplished by a novel encoding of Turing machines by means of memory bounded protocols.

The second contribution of this paper is on the formalization of Progressing Collaborative Systems that may create fresh values. We introduced the notion of progressing in [11]. Progressing collaborative systems were inspired by protocols, namely, on the fact that a protocol session is always progressing. That is, once one step of a protocol session is taken, the same step is no longer repeated. In [11], we formalized this notion in the context of Collaborative Systems [14], such as administrative processes. Administrative systems normally also have this progressing nature: once an item in an activity to-do list is checked, that activity is not repeated.

However, in [11], we limited ourselves to systems that did not create fresh values, such as nonces. Combining the progressing condition with the creation of fresh values turned out to be surprisingly challenging because of a subtle interaction between the two features. We discuss this in detail in Section 4. This paper extends the formalization of Progressing in [11] to systems that may create fresh values, based on the machinery introduced in [10]. We also prove that the reachability problem for Progressing Systems that may create fresh values in NP-complete.

This paper is structured as follows:

- Section 2 reviews the specification of bounded memory protocols, the Dolev-Yao Adversary, and of Bounded Memory Adversaries. It also reviews some of the complexity results for the secrecy problem;

- Section 3 contains the undecidability proof with memory bounded protocols. This is a novel, stronger undecidability proof, which allows us to infer that it is not possible to determine an upper bound on the memory of the Dolev-Yao adversary from the memory bound of the protocol;
- Section 4 contains the formalization of Progressing Collaborative System that may create nonces. We argue that its precise formalization is only meaningful when bounding the memory of the participants of the system. We also prove the NP-completeness of the reachability problem;

Finally in Sections 5 and 6 we comment on related work and conclude by pointing out to future work.

2 Bounded Memory Protocols and Adversaries

We formalize bounded memory protocol theories and adversary theories by means of multiset rewrite rules, similarly as in [7]. Assume fixed a sorted first-order alphabet consisting of constant symbols, $c_1, c_2, \ldots$, function symbols, $f_1, f_2, \ldots$, and predicate symbols, $P_1, P_2, \ldots$ all with specific sorts (or types). The multi-sorted terms over the signature are expressions formed by applying functions to arguments of the correct sort. A *fact* is a ground, atomic formula over multi-sorted terms. Facts have the form $P(t_1, \ldots, t_n)$ where P is an n-ary predicate symbol, where $t_1, \ldots, t_n$ are terms, each with its own sort. A *state*, or *configuration* of the system is a finite multiset W of facts.

The *size of a fact* is the total number of term and predicate symbols it contains. We count one for each predicate, function, constant, and variable symbols. We use $|F|$ to denote the size of a fact F. For example, $|P(x, c)| = 3$, and $|P(f(z, x, n), z)| = 6$. We will normally assume in this paper an upper bound on the size of facts, as in [7, 14].

A state, or configuration of the system is a finite multiset of grounded facts, *i.e.*, facts that do not contain variables. Configurations, intuitively, specify the state of the world and are modified by actions. In general, an action is a multiset rewrite rule of the following form:

$$X_1, \ldots, X_n \longrightarrow \exists \boldsymbol{x}. Y_1, \ldots, Y_m \tag{1}$$

where the X_is and Y_js are facts. The collection $X_1, \ldots, X_n$ is called the pre-condition of the rule, while $Y_1, \ldots, Y_m$ is called post-condition. We assume that all free variables are universally quantified. By applying the action for a ground substitution (σ), the pre-condition applied to this substitution ($X_1\sigma, \ldots, X_n\sigma$) is replaced with the post-condition applied to the same substitution ($Y_1\sigma, \ldots, Y_m\sigma$). In this process, the existentially quantified variables ($\boldsymbol{x}$) appearing in the post-condition are replaced by fresh variables, also called nonces in protocol security literature. The rest of the configuration remains untouched. Thus, we can apply the action $P(x), Q(y) \rightarrow_A \exists z.R(x, z), Q(y)$ to the global configuration $V, P(t), Q(s)$ to get the global configuration $V, R(t, c), Q(s)$, where the constant c is new.

Given a multiset rewrite system R, one is often interested in the *reachability problem*: Is there a sequence of (0 or more) rules from R which transforms configuration W into Z? If this is the case then we say that Z is reachable from W using R.

Balanced Actions and Empty Facts An important condition for formalizing balanced protocols is that of *balanced actions*. Balanced actions were introduced in the context of collaborative systems [14]. We classify an action as *balanced* if the number of facts in its precondition is the same as the number of facts in its postcondition. That is, $n = m$ in Equation 1. If we restrict all actions in a system to be balanced, then the size of the configurations in a run remains the same as in the initial configuration. Since we assume facts to have a bounded size, the use of balanced actions imposes a bound on the storage capacity of the agents, *i.e.*, balanced systems have constant memory. Creating a new fact by means of a balanced action amounts to inserting that fact into the resulting configuration by replacing a fact appearing in the enabling configuration. In other words the memory of the system is only updated. No new memory space is created.

In order to support the creation of new facts in balanced systems, we use *empty facts*, written $P(*)$. Intuitively, an empty fact denotes an available memory slot that could be filled by some new information. Here $*$ is not a constant, but just used for illustrative purposes. By using empty facts, one can transform unbalanced systems into balanced systems simply by adding enough empty facts to the precondition or the postcondition of each rule with so that it becomes balanced.

2.1 Bounded Memory Protocols

A bounded memory protocol, formally defined below, only contains balanced actions [10]. This means that the number of facts known by the participants at a given time is bounded. Bounding the memory available for protocol sessions also intuitively bounds the number of concurrent protocol sessions. This is because for each protocol session, one needs some free memory slots to remember, for instance, the internal states of the agents involved in the session. However, this does not mean that there may be an unbounded number of protocol sessions in a trace. Once a protocol session is completed, the memory slots it required can be re-used to initiate a new protocol session.

This is different to the well-founded protocol theories in [7] where the rules are not necessarily balanced and where all protocol sessions are created at the beginning of the trace before any protocol session starts executing. In well-founded protocol theories, an unbounded number of protocol sessions can run concurrently and therefore participants are allowed to remember an unbounded number of facts.

Definition 1. *A theory $\mathcal{A}$ is a* balanced role theory *if there is a finite list of predicate names called the* role states $S_0, S_1, \ldots, S_m$ *for some m, such that every rule $L \to \exists \boldsymbol{t}.R$ in $\mathcal{A}$ is balanced and there is exactly one occurrence of a state predicate in L, say S_i, and exactly one occurrence of a state predicate in R, say S_j, such that $i < j$. We call the first role state, S_0,* initial role state*, and the last role state S_m* final role state. *Only rules with final role states can have an empty fact in the post-condition.*

Defining roles in this way, ensures that each application of a rule in $\mathcal{A}$ advances the state forward. Each instance of a role can only result in a finite number of steps in a trace. The request on empty facts formalizes the fact that one of the participants, either the initiator or the responder, sends the "last" protocol message. In [10], one can find several examples of protocols specified as balanced role theories.

In order to allow for an unbounded number of protocol sessions in a trace, we allow protocol roles to be created at any time with the of cost of consuming empty facts $P(*)$. At the same time, we allow protocol sessions that have been completed to be forgotten. That is, once its final role state has been reached, it can be deleted, creating in the process new empty facts $P(*)$. These empty facts can then be used to create new protocol roles starting hence a new protocol session. Such theories are called role regeneration theories.

Definition 2. *If $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are balanced role theories, a* role regeneration theory *is a set of rules that either have the form*
$Q_1(\boldsymbol{x}_1) \cdots Q_n(\boldsymbol{x}_n) P(*) \to Q_1(\boldsymbol{x}_1) \cdots Q_n(\boldsymbol{x}_n) S_0(\boldsymbol{x})\,,$
where $Q_1(\boldsymbol{x}_1) \ldots Q_n(\boldsymbol{x}_n)$ is a finite list of facts not involving any role states, and S_0 is the initial role state for one of theories $\mathcal{A}_1, \ldots, \mathcal{A}_k$, or the form
$S_m \to P(*)\,,$
where S_m is the final state for one of theories $\mathcal{A}_1, \ldots, \mathcal{A}_k$.

The explicit use of balanced actions in role theories and role regeneration theories allows us to bound the memory of the participants, including the number of concurrent protocols in the system, without bounding the total number of protocol sessions in a trace.

Definition 3. *A pair $(\mathcal{P}, H)$ is a* bounded memory protocol theory *if H is a finite set of facts (called* initial set*), and $\mathcal{P} = \mathcal{R} \uplus \mathcal{A}_1 \uplus \cdots \uplus \mathcal{A}_n$ is a protocol theory where $\mathcal{R}$ is a role regeneration theory involving only facts from H and the initial and final roles states of $\mathcal{A}_1, \ldots, \mathcal{A}_n$, and $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are balanced role theories. For role theories $\mathcal{A}_i$ and $\mathcal{A}_j$, with $i \neq j$, no role state predicate that occurs in $\mathcal{A}_i$ can occur in $\mathcal{A}_j$.*

Intuitively, a bounded memory protocol theory specifies a particular scenario to be model-checked involving some given protocol(s). Besides empty facts, $P(*)$, the finite initial set of facts contains all the facts with the information necessary to start protocol sessions, for instance, shared and private keys, the names of the participants of the network, as well as any compromised keys. Here, for simplicity we assume only symmetric keys, although other types of keys can be also formalized.

2.2 Standard Dolev-Yao and Bounded Memory Dolev-Yao Adversaries

The powerful adversary proposed by Dolev-Yao [6] acts as the network, that is, all messages communicated are sent through the adversary. He hears everything and learns messages modulo encryption. More precisely, he is capable of intercepting any message sent by a protocol participant and then store the received information, decompose it and decrypt with the keys he possesses. He cannot, however, decrypt messages for which he does not have the correct key. Moreover, he can also create fresh values, encrypt, compose messages from the information he has learned. One of his major strengths is that he can remember as much information as he wants, *i.e.*, his memory is unbounded.

Figure 1a. depicts the rules of such an adversary. The I/O rules specify the fact that the adversary acts as the network receiving all messages sent (N_S) and sending all messages that are received (N_R). The remaining rules are straightforward, specifying

I/O Rules:

REC :	$N_S(x) \to M(x)$
SND :	$M(x) \to N_R(x)$
	Decomposition Rules:
DCMP :	$M(\langle x, y\rangle) \ \to M(x)\ M(y)$
DECS :	$M(k)\ M(enc(k, x)) \to$ $M(k)\ M(enc(k, x))\ M(x)$
	Composition Rules:
COMP :	$M(x)\ M(y) \to M(\langle x, y\rangle)$
USE :	$M(x) \to M(x)\ M(x)$
ENCS :	$M(k)\ M(x) \to$ $M(k)\ M(enc(k, x))$
GEN :	$\to \exists n.M(n)$

a. Theory for the Standard Dolev-Yao Adversary

I/O Rules:

REC:	$N_S(x) \to M(x)$
SND:	$M(x) \to N_R(x)$
	Decomposition Rules:
DCMP:	$M(\langle x, y\rangle)\ P(*) \to M(x)\ M(y)$
DEC:	$M(k)\ M(enc(k, x))\ P(*)$ $\to M(k) M(x)\ M(enc(k, x))$
	Composition Rules:
COMP:	$M(x)\ M(y) \to M(\langle x, y\rangle)\ P(*)$
USE:	$M(x) P(*) \to M(x) M(x)$
ENC:	$M(k) M(x) \to M(k)\ M(enc(k, x))$
GEN:	$P(*) \to \exists n.M(n)$
	Memory maintenance rule:
DELM:	$M(x) \to P(*)$

b. Bounded Memory Dolev-Yao Adversary Theory

Fig. 1: Theories for the Standard and the Bounded Memory Adversaries

when the adversary may decompose and compose messages. Notice that contrary to the formalization of the bounded memory protocols, the actions specifying the Dolev-Yao adversary are not all balanced. In particular, the adversary may always learn new facts, such as in the actions DECS and GEN, where the adversary learns the contents of an encrypted message and creates a nonce.

In [10], we proposed a Bounded Memory Dolev-Yao adversary, which has many capabilities of the Dolev-Yao adversary. He can intercept, send and compose messages, create nonces, etc. But differently from the Dolev-Yao adversary, he can remember only a bounded number of facts at a given time. This is formally imposed by the balanced adversary theory presented in Figure 1b. In order for him to store some new information, such as a nonce, he might have to forget some information he previously learned. This is specified by additional memory maintenance rule.

2.3 Complexity Results for the Secrecy Problem

In an interaction of malicious adversaries with honest participants, one is interested in *secrecy problem*, namely, in determining whether the adversary can discover a secret s. Formally it is an instance of the reachability problem: Is it the case that a configuration containing $M(s)$, where s is a secret originally owned by an honest participant can be reached from an initial configuration?

Undecidability of the Secrecy Problem It is known for some time that the secrecy problem is undecidable in general [7]. The undecidability proof in [7] proceeds by encoding the existential Horn implication problem, which is also proved to be undecidable. However, in that work, one used *well-founded protocol theories*, where the memory of the protocol is unbounded. For instance, in *well-founded protocol theories*, it is allowed for an unbounded number of concurrent protocol sessions to run at the same time. In fact, all the protocol sessions in a trace are initialized at the beginning before any session starts. This implies that the participants of the system may remember an unbounded number of facts, namely, the facts containing the information of the protocols in which they are participating in.

In Section 3, we strengthen the result in [7], by showing that the secrecy problem is undecidable *even if the memory of the protocol is bounded.* This is accomplished by a novel encoding of Turing machines by means of memory bounded protocols.

PSPACE-completeness of the Secrecy problem for the Bounded Memory Dolev-Yao Adversary Besides proposing the Bounded Memory Dolev-Yao Adversary and demonstrating that he can carry out known anomalies when given enough memory, we proved in [10] that the secrecy problem when assuming the Bounded Memory Dolev-Yao Adversary is PSPACE-complete. The key insight for this result was showing how to handle the fact that a trace may have an exponential number of nonces, which seems to preclude PSPACE membership. We circumvent the problem of requiring too many fresh values in a trace by reusing obsolete constants instead of creating new values.

The argument goes roughly like this: since all actions of the system are balanced, including those specifying the adversary (see Figure 1b), the number of facts in any configuration remains the same as in the initial configuration, namely m. Moreover, as we assume an upper bound on the size of facts, namely k, then any configuration in a trace has at most mk symbols. We can then fix a priori a polynomial number of nonce names, namely, $2mk$ names, so that whenever one needs a fresh nonce, one can find a name in this set of $2mk$ names that is fresh to the participants. It may well be the case that some name in this set of nonce names is used many times in a trace. However, for the participants at that point of the trace, the name used seems fresh as no participant can remember it.

This idea will be key for our proposal of Progressing systems with nonce generation in Section 4.

3 Protocol security is very undecidable: A bound on the adversary cannot be inferred from a bound on a protocol

We now detail the sound and faithful encoding of Turing machines using bounded memory protocols.

3.1 Encoding of Turing Machine Tapes

Without loss of generality, let $\mathcal{M}$ be a Turing machine such that

(i) $\mathcal{M}$ has only one tape, which is one-way unbounded to the right. The leftmost cell (numbered by 0) contains the marker \$ unerased;

(ii) The initial 3-cell configuration is of the following form, where B stands for the blank symbol:

$$\begin{array}{|c|c|c|}\hline \$ & \langle q_1, B\rangle & B \\ \hline \end{array} \tag{2}$$

We write $\boxed{\;\langle q, \xi\rangle\;}$ to denote that the corresponding cell contains the symbol ξ and is scanned by $\mathcal{M}$ in its state q.

(iii) We assume that all instructions are "move" instructions. The head of $\mathcal{M}$ cannot move to the leftmost cell marked with \$.
(iv) Finally, $\mathcal{M}$ has only one *accepting* state, q_0.

Encoding of the Tape In our encoding, we need two honest participants only, Alice and Bob. Assume they share a symmetric key K, not known to any other participant. We will encode the tape cells separately as follows:

(a) An unscanned cell that contains symbol ξ_0 is encoded by a term encrypted with the key K
$E_K(\langle t_0, \xi_0, e_0, t_1\rangle)$,
where t_0 and t_1 are nonces, and $e_0 = 1$ if the cell is the last cell in a configuration.
(b) The cell that contains symbol ξ and is scanned by $\mathcal{M}$ in state q is also encoded by a term encrypted with the key K
$E_K(\langle t_1, \langle q, \xi\rangle, 0, t_2\rangle)$
where t_1 and t_2 are nonces.

Motivation: The nonces t_0 and t_1 in the terms encoding the tape cell are used for two purposes:

(a) Firstly, t_0 and t_1 are served as "timestamps" for the a visit by $\mathcal{M}$ in the cell. Whenever $\mathcal{M}$ re-visits this cell, the previous term is updated with fresh nonces indicating a new visit;
(b) Secondly, as t_0 and t_1 are unique, they are used to uniquely link cells that are adjacent to each other.

For example, the initial configuration, Equation (2), with three cells is encoded by using the sequence of nonces t_0, t_1, t_2, t_3 as shown below:
$\langle E_K(\langle t_0, \$, 0, t_1\rangle), E_K(\langle t_1, \langle q_1, B\rangle, 0, t_2\rangle), E_K(\langle t_2, B, 1, t_3\rangle)\rangle$
Notice the role of the nonces t_0, t_1, t_2, t_3. For instance, the nonce t_1 is used to correctly encode the fact that the cell $\langle q_1, B\rangle$ is to the right of the cell with the mark \$.

3.2 Encoding Turing Machine's Actions as a Bounded Memory Protocol

Given a Turing machine $\mathcal{M}$ and the encoding of tapes discussed above, we encode its actions by means of bounded memory protocol called $\mathcal{P}_{\mathcal{M}}$. We describe the role of Alice (initiator) and Bob (responder):

Alice's Role Assume that Alice is the initiator and her initial state is:
$\langle E_K(\langle t_0, \$, 0, t_1\rangle), E_K(\langle t_1, \langle q, B\rangle, 0, t_2\rangle), E_K(\langle t_2, B, 1, t_3\rangle), E_K(\langle t_4, B, 1, t_5\rangle)\rangle$
The protocol starts by Alice updating all nonces t_i to t_i', and sending the following updated message to Bob. At this point, she does not need to remember the previous terms using the nonces t_i. Notice that the last term does not share nonces with the first three. It will be used for extending the tape.
$\langle E_K(\langle t_0', \$, 0, t_1'\rangle), E_K(\langle t_1', \langle q, B\rangle, 0, t_2'\rangle), E_K(\langle t_2', B, 1, t_3'\rangle), E_K(\langle t_4', B, 1, t_5'\rangle)\rangle$
That is she erases her memory and is ready to store new facts. In particular, she is waiting for a message from Bob of the form:

$\langle E_K(\langle t_0, \alpha_0, 0, t_1\rangle), E_K(\langle \widetilde{t}_1, \alpha_1, 0, \widetilde{t}_2\rangle), E_K(\langle t_2, \alpha_2, e_2, t_3\rangle), E_K(\langle t_4, B, 1, t_5\rangle)\rangle$
By verifying its integrity with $(t_1 = \widetilde{t}_1)$ and $(\widetilde{t}_2 = t_2)$, Alice assumes that there is no intrusion in the channel. If some α_i is of the form $\langle q_0, \xi\rangle$, then Alice sends openly a *secret* to Bob, otherwise, Alice sends a neutral message.

Bob's role The role of Bob is to transform the message received with the help of an instruction from the given Turing machine $\mathcal{M}$. Bob is expecting to receive a message (presumably from Alice) of the form:
$\langle E_K(\langle t_0, \xi_0, 0, t_1\rangle), E_K(\langle \widetilde{t}_1, \langle q, \xi\rangle, 0, \widetilde{t}_2\rangle), E_K(\langle t_2, \xi_2, e_2, t_3\rangle), E_K(\langle t_4, B, 1, t_5\rangle)\rangle$
Bob verifies its integrity by $(t_1 = \widetilde{t}_1)$ and $(\widetilde{t}_2 = t_2)$, and follows one of three cases:

(1) (Extending the tape) For $e_2 = 1$, Bob *updates* nonces t_i to t'_i, and sends the following updated message to Alice, which provides a new last cell in the chain of four cells
$\langle E_K(\langle t_0, \xi_0, 0, t'_1\rangle), E_K(\langle t'_1, \langle q, \xi\rangle, 0, t'_2\rangle), E_K(\langle t'_2, \xi_2, 0, t'_3\rangle), E_K(\langle t'_3, B, 1, t'_4\rangle)\rangle$

(2) (Moving the Head of the Machine to the Right) For an $\mathcal{M}$'s instruction of the form $q\xi \to q'\eta R$, denoting: *"if in state q looking at symbol ξ, replace it by η, move the tape head one cell to the right, and go into state q'"*, Bob *updates* some nonces t_i to t'_i, and sends the following updated message to Alice
$\langle E_K(\langle t_0, \xi_0, 0, t'_1\rangle), E_K(\langle t'_1, \eta, 0, t'_2\rangle), E_K(\langle t'_2, \langle q', \xi_2\rangle, 0, t_3\rangle), E_K(\langle t_4, B, 1, t_5\rangle)\rangle$

(3) (Moving the Head of the Machine to the Left) For an $\mathcal{M}$'s instruction of the form $q\xi \to q'\eta L$, denoting: *"if in state q looking at symbol ξ, replace it by η, move the tape head one cell to the left, and go into state q'"*, Bob *updates* some nonces t_i to t'_i, and sends the following updated message to Alice
$\langle E_K(\langle t_0, \langle q', \xi_0\rangle, 0, t'_1\rangle), E_K(\langle t'_1, \eta, 0, t'_2\rangle), E_K(\langle t'_2, \xi_2, 0, t_3\rangle), E_K(\langle t_4, B, 1, t_5\rangle)\rangle$

Remark 1. Both Alice and Bob can input and output only messages of the form
$\langle E_K(\langle t_0, \alpha_0, 0, t_1\rangle), E_K(\langle t_1, \alpha_1, 0, t_2\rangle), E_K(\langle t_2, \alpha_2, e_2, t_3\rangle), E_K(\langle t_4, B, 1, t_5\rangle)\rangle$
where the first three components represent the chain of three cells, and the fourth component refers to the last cell in a configuration.

Remark 2. The above protocol is balanced. It can be formalized by a bounded memory protocol. In fact, we have formalized it in Appendix. In particular, only terms of height fixed in advance are used. Nonces are only updated, that is, the old nonces are replaced by new nonces. Therefore, Alice and Bob can forget the old nonces. In fact, Alice and Bob are finite automata, which are allowed to *update nonces* only.

3.3 A Man-in-the-Middle Attack by Mallory

Notice that, according to Remark 1, by active eavesdropping Mallory can accumulate terms of the form

$$E_K(\langle t_1, \alpha_1, e_1, t_2\rangle) \tag{3}$$

if and only if they are components of outputs generated by Alice or by Bob. We now discuss the following attack on the protocol above:

(1) For the first run, Mallory intercepts the initial message from Alice, stores it, and resends it to Bob. While Bob responds, Mallory intercepts the message from Bob, stores it, and resends it to Alice.

(2) For each of the next runs, Mallory first intercepts the initial message from Alice. Taking non-deterministically terms of the form (3) from his memory, Mallory then composes a message of the form:

$$\langle E_K(\langle t_0, \alpha_0, 0, t_1\rangle), E_K(\langle \widetilde{t_1}, \alpha_1, 0, \widetilde{t_2}\rangle), E_K(\langle t_2, \alpha_2, e_2, t_3\rangle), E_K(\langle t_4, B, 1, t_5\rangle)\rangle$$

and sends it to Bob. If Bob accepts this message and responds with a transformed one as described in the protocol, then Mallory intercepts this new message from Bob, stores it, and resends it to Alice.

The following lemma shows a certain chain-like structure of the terms accumulated by the adversary. These chain-like structure are specified by the use of nonces and each chain corresponds to reachable configurations of the Machine $\mathcal{M}$.

Lemma 1. *Suppose that a term of the form $E_K(\langle t, \langle q, \xi\rangle, 0, t'\rangle)$ appears in the intruder memory by active eavesdropping. Then there is a unique sequence of nonces t_0, t_1,..., t_{n+2} and a chain of terms from the adversary's memory*

$$\begin{array}{l} E_K(\langle t_0, \$, 0, t_1\rangle),\ E_K(\langle t_1, x_1, 0, t_2\rangle), \dots .\ E_K(\langle t_{j-1}, x_{j-1}, 0, t_j\rangle), \\ E_K(\langle t_j, \langle q, x_j\rangle, 0, t_{j+1}\rangle),\ E_K(\langle t_{j+1}, x_{j+1}, 0, t_{j+2}\rangle), \dots,\ E_K(\langle t_n, x_n, 0, t_{n+1}\rangle), \\ E_K(\langle t_{n+1}, B, 1, t_{n+2}\rangle) \end{array}$$

such that

(a) $t_j = t$, $x_j = \xi$, *and* $t_{j+1} = t'$,

(b) *$\mathcal{M}$ leads from the empty initial configuration to the configuration where the string $x_1 x_2..x_j..x_n$, is written in cells* 1, 2,..,*j*,..,*n on the tape*

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|}\hline \$ & x_1 & x_2 & \cdot & \cdot & x_j & \cdot & \cdot & x_n & \\ \hline \end{array}\cdots$$

and the j-th cell is scanned by $\mathcal{M}$ in state q.

Proof. By induction on the number of actions performed by Bob to outcome a message one of the components of which is $E_K(\langle t, \langle q, \xi\rangle, 0, t'\rangle)$. ■

Theorem 1. *There is a Dolev-Yao attack on the above protocol if and only if the machine $\mathcal{M}$ terminates on the empty input.*

Proof. We sketch the proof of both directions of the proof.

(a) The direction from a terminating computation to an attack is straightforward by induction on the length of the computation.

(b) The inverse direction is quite tricky. In the case of a successful attack, a term of the form $E_K(\langle \widetilde{t_1}, \langle q_0, \xi\rangle, 0, \widetilde{t_2}\rangle)$, must appear in the adversary's memory. Then by Lemma 1, $\mathcal{M}$ leads from the empty initial configuration to a final configuration where a cell is scanned in state q_0. ■

Notice that in all attacks above the attacker in fact does not need to create/update fresh nonces, but simply intercept, decompose, compose and copy messages.

Corollary 1. *The existence of a Dolev-Yao attack is undecidable even for bounded memory protocols, $\mathcal{P}_{\mathcal{M}}$, where Alice and Bob are finite automata whom are allowed to update nonces only, all actions by Alice and Bob are balanced, and only terms of height fixed in advance are used by Alice, Bob, and an adversary (even if the actions of the adversary are limited to decompose, compose, and copy).*

Proof. Given a non-recursive recursively enumerable set S, and a sequence of Turing machines $\mathcal{M}_n$ such that $\mathcal{M}_n$ terminates on the empty input iff $n \in S$, it suffices to consider the corresponding bounded memory protocols $\mathcal{P}_{\mathcal{M}_n}$. ■

Thus an upper bound on the memory of the Dolev-Yao adversary is not computable from a bound on the memory used by a protocol. Based on peculiarities of our encoding described in Section 3.2, we can express such a phenomenon in quantitative terms.

Theorem 2. *Whatever a total recursive function h we take, we can construct a recursive sequence of bounded memory protocols $\mathcal{Q}_n$ so that*

(a) *For any n, there is a Dolev-Yao attack on the bounded memory protocol $\mathcal{Q}_n$.*
(b) *However, for any n starting from some n_0, any Dolev-Yao adversary the size of whose memory is bounded by $h(n)$ is not capable of detecting an attack on the bounded memory protocol $\mathcal{Q}_n$.*

Proof Sketch. Given a total recursive recursive function f, as $\mathcal{Q}_n$ we take the bounded memory protocol $\mathcal{P}_{M_n}$ described in Section 3.2, where M_n is a Turing machine terminating on the empty input with the value $f(n)$.

Roughly, according to Theorem 1, Mallory, whose memory size is bounded by $h(n)$, can play at most $2^{O(h(n))}$ steps. It suffices, therefore, to take the function f such that its time complexity is $\Omega(2^{2^{h(n)}})$. ■

4 Progressing Collaborative Systems with Fresh Values

We introduced the notion of progressing in [11] in the context of Collaborative Systems. Progressing is inspired by the nature of security protocols, as well as many administrative and business processes. Namely, once one step of a protocol session is taken, the same step is not repeated. Similarly, whenever one initiates some administrative task, one receives a "to-do" list with the activities or tasks that have to be performed or achieved. Once an item on the list has been "checked", one does not need to return to this item anymore. When all the items have been checked, the process ends. Such a process is always advancing. Consequently, protocols and administrative processes have a polynomial number of steps, in other words they can be considered as *efficient*. That is, one does not need to perform an exponential number of actions to conclude such processes.

To formally capture this intuition, we defined Progressing in [11] as follows: A sequence of actions is *progressing* if *an instance* of an action appears at most once. This notion of progressing reflects the requirement that progressing processes are efficient, as one needs to consider only traces of polynomial length to check whether a process can be completed or not. For instance, the Towers of Hanoi problem has no progressing plans, since any plan is of exponential length, which implies that one and the same action is necessarily used an exponential number of times. In fact, the progressing reachability problem for systems that do not create nonces is NP-complete [11].

However, extending this notion of progressing to systems that can create nonces turned out to be quite challenging. The problem arises from the fact that if we allow actions to create fresh values, one may capture processes which require an exponential number of actions, that is, processes that cannot be efficiently carried out. Let us try to extend *naively* the progressing definition above to the case when actions may create nonces as follows: A plan with nonces is progressing if an instance of an action, with the same constants and the same nonces, appears at most once. Unfortunately, such a definition of progressing is not satisfactory. When a nonce is created, it is fresh, meaning that it hasn't appeared in the system as yet. Consequently, every application of an action that creates a nonce is a new instance of that action. For instance, we can adapt the encoding of the Towers of Hanoi, so that for each move creates a new nonce. Thus each action is a different instance, because a different nonce is used and created. Therefore, the Towers of Hanoi would be according to the naive definition above progressing, which is clearly not what we want.

Therefore, in order to extend the notion of progressing to the case where actions may create nonces, we shouldn't allow unbounded nonce generation. Instead we need to somehow limit the use of nonces, but how many nonces is enough? This question is answered for the case when systems are balanced. As discussed in Section 2.3, for the case of *balanced systems*: one can simulate any plan that uses an unbounded number of nonces by fixing a priori a polynomial number of nonce names [10] with respect to the number of facts in the initial configuration (m) and the upper-bound on the size of facts (k). In the following sections, we formalize these intuitions.

4.1 Balanced Progressing with Fresh Values

We extend the notion of progressing for balanced systems that can create fresh values. Central to our notion will be the definition of when two instances of actions are equivalent (Definition 4). Consider for example the following two instances of an action, where the t_is are terms and n_js are nonce names which do not appear in the alphabet of the language:

$$X_1(t_1)X_2(t_2, t_3, n_1)X_3(n_1, n_2) \to \exists x.X_4(t_1)X_2(t_2, x, n_3)X_5(n_1, n_3),$$
$$X_1(t_1)X_2(t_2, t_3, n_4)X_3(n_4, n_5) \to \exists x.X_4(t_1)X_2(t_2, x, n_6)X_5(n_4, n_6).$$

These instances only differ in the nonce names used: the same fresh value, n_3 in the former instance and n_6 in the latter, appear in same facts exactly at the same places, and similarly, for the pairs of nonces (n_1, n_4), and (n_2, n_5). Inspired by a similar notion in λ-calculus [3], and α-equivalence among configurations in [10], we regard instances of actions that differ only in the nonce's names used, as equivalent.

Definition 4. *Two* instances *of an action, r_1 and r_2, are* equivalent *if there is a bijection σ that maps the set of all nonce names appearing in one instance to the set of all nonce names appearing in the other instance, such that $r_1\sigma = r_2$.*

The two instances given above are equivalent because of the following bijection $\{(n_1, n_4), (n_2, n_5), (n_3, n_6)\}$. It is easy to show that the above relation among instances of actions is indeed an equivalence relation.

Definition 5. *Given a balanced multiset rewrite system R, an initial configuration W and a polynomial $f(m, k)$, we say that a sequence of actions is* progressing *if it contains at most $f(m, k)$ equivalent instances of any action, where m is the number of facts in the configuration W and k is the upper bound on size of facts.*

Progressing reachability problem has a solution if for a given multiset rewrite system R and configurations W and Z, there is a *progressing* sequence of actions from R which transforms configuration W into Z.

Notice that our new notion of progressing extends progressing from [11], as they coincide when systems do not allow fresh values. We will, therefore, be able to compare our complexity results.

Furthermore, as per Definition 5, not every computation could be considered as progressing. Here a nonce name may only be used by the same action a polynomial number of times in a computation. Hence, not every reachability problem that has a solution will have a progressing solution. For example, in any solution of Towers of Hanoi puzzle, one and the same nonce name has to be updated an exponential number of times by the only action from the representation of this puzzle in [10]. Therefore this problem has no progressing solution as per our Definition 5.

Notice that, if nonces are allowed, we only conceive progressing in balanced systems, while progressing with no nonces is clear for any in any multiset rewriting system, even the unbalanced ones. This is because nonce update from [10] was only possible in balanced systems.

4.2 Complexity Results for Progressing Systems with Fresh Values

We now investigate the progressing reachability problem when actions can create fresh values.

Theorem 3. *Given a multiset rewrite system R with only balanced actions that can create fresh values, an initial and a final configurations, an upper-bound, k, on the size of facts and a polynomial f with two parameters, the progressing reachability problem is NP-complete.*

Proof We infer the NP lower bound from the encoding of the 3-SAT problem from [11], which is well-known to be NP-complete [5].

For the NP upper bound, Assume given an initial configuration with m facts and a polynomial f with two parameters. Moreover, let n be the number of rules in R, d is the number of constant and function symbols, k the upper bound on the size of facts and l the upper bound on the number of different variables appearing in a rule in R. Here we assume k and l to be much smaller than d and m.

Following [10], we can assume that all nonces are used from a $2mk$ set of nonce names, that is fixed a priori. Hence, the number of constants in the system is $d{+}2mk$. As actions are applied, instead of fresh values being created, nonces are updated. Obsolete nonce names are picked from the fixed set of $2mk$ nonce names. They are, therefore, different form any nonce in the configuration and can be considered fresh.

Since the size of facts is bounded, we do not need to consider terms that have a size greater than k. Therefore we need to consider at most $(d + 2mk)^k$ terms. Since in progressing traces, one is allowed to use only a polynomial number of instances of a rule, the length of traces is bounded by

$$f(m, k) \times n \times \left((d + 2mk)^k\right)^l = f(m, k) \times n \times (d + 2mk)^{kl} \, ,$$

The above bound is therefore polynomial in the size of the configurations, number of rules and symbols.

Assume that W is the initial configuration and Z is the goal configuration that is the configuration one wants to reach. Also assume that one can check in polynomial time whether a configuration is the final one or not. We show below that there is a polynomial-time deterministic algorithm that checks for valid computations.

We show that we can check in polynomial time, where a plan solves the progressing problem. Let S_i be the configuration at step i, so $S_0 = W$, Q_i be the multiset of pairs, $\langle r, \sigma \rangle$, of rules and substitutions used before step i, so $Q_0 = \emptyset$.

1. Check if $Z \subseteq S_{i-1}$, then ACCEPT; otherwise continue;
2. Guess an action $r_i : X_i \to Y_i$, and a substitution σ_i;
3. Check if $X_i\sigma_i \in S_{i-1}$, then continue; otherwise FAIL;
4. Check if the multiplicity of $\langle r_i, \sigma_i \rangle$ in Q_{i-1} is greater than $f(m, k)$, then FAIL; otherwise continue;
5. $S_i = S_{i-1} \cup \{Y\sigma_i\} \setminus \{X\sigma_i\}$;
6. $Q_i = Q_{i-1} \cup \{\langle r_i, \sigma_i \rangle\}$;
7. Increment i.

Since the size of facts is bounded, all steps are done in polynomial time. The only step that may not be apparent is step 4. However, the set Q_i is bounded by the length of the computation. Therefore, the reachability problem is in NP. □

Although we prove the NP-completeness above for the reachability problem, the result above can be easily be extended to the other compliance problems detailed in [12].

5 Related Work

This paper strengthens the undecidability proof given in [7]. In particular, the proof in [7] uses an encoding with well-founded protocols theories, whereas our proof uses an encoding with bounded memory protocols. In well-founded protocols, it is possible for the honest participants to have an unbounded memory, while in bounded memory protocols the memory of the honest participants is bounded. This is in fact the case in the undecidability proof given in [7]. The proof relies on an unbounded number of protocol sessions. Moreover, all these protocols sessions are created before any sessions starts executing, the participants require an unbounded memory to remember in which protocol sessions they are participating. On the other hand, in our proof, Alice and Bob participate in one protocol session at a time. Whenever one is finished, they can re-use their memory to participate in the subsequent protocol session. This difference is crucial, as with our proof, we can infer that there is no way to compute an upper bound on the memory of the adversary from the memory bounds of the participants, demonstrating further the hardness of the secrecy problem.

Our NP upper bound for the progressing reachability problem in Theorem 3 is different from the NP upper bound obtained [1, 18] in the context of protocol security. In their models, the progressing condition is incorporated syntactically into the rules of the theories. Specifically, they use role predicates of the form A_i contain an index i denoting the stage in the protocol. The NP-completeness result in [1, 18] is obtained by further restricting systems to have only a bounded number of roles. We, on the other hand, bound the number of instances of actions that can appear in a plan. It would be interesting, however, to check whether our assumption on the existence of an upper-bound on the size of facts could be relaxed as in [18].

Harrison *et al.* present a formal approach to access control [9] and faithfully encode a Turing machine in their system. However, in contrast to our encoding, they use a non-commutative matrix to encode the sequential, non-commutative tape of a Turing machine. In their proofs, the non-commutative nature of the encoding plays an important role. We, on the other hand, encode Turing machine tapes by using commutative multisets. Specifically, they show that if no restrictions are imposed to the systems, the reachability problem is undecidable.

Much work on reachability related problems has been done within the Petri nets community, see *e.g.*, [8]. Specifically, we are interested in the *coverability problem* which is closely related to the reachability problem in multiset rewrite systems. To the best of our knowledge, no work that captures exactly the balanced condition nor the progressing definition with nonce creation has yet been proposed. In these cases, it does not seem possible to provide direct, *faithful* reductions between our systems and Petri nets.

6 Conclusions

This paper showed that the memory of the adversary cannot be inferred from the memory bounds of the participants. This is accomplished by proposing a novel undecidability proof by encoding Turing machines by means of bounded memory protocols. This result confirms the hardness of protocol security. This answers negatively an open problem left in [10]. Our second contribution was on the formalization of progressing for balanced systems that can create fresh values. We believe that this fragment will provide the foundations for a useful class of systems, namely for systems such as adminstrative processes where the same instance of an action should not be performed an exponential number of times. Finally, we proved the NP-completeness for the Progressing reachability problem.

There are many directions to investigate from here. For instance, it would be interesting to check whether one can adapt the encoding of the Horn implication problem given in in [7] to use bounded memory protocols, instead of well-founded ones. Another direction is whether one can improve the NP-completeness proof by relaxing the assumption on the upper-bound of facts. This was possible in the context of protocol security as shown in [18].

We are currently colaborating with Carolyn Talcott on the use of the computational tool Maude [4] for the specification and model-checking of regulated processes, such as administrative processes [13]. In particular, we are investigating on whether our NP-completeness proof can improve Maude's performance in model-checking Progressing systems.

Another direction that we are currently investigating is to extend our model with real times. In particular, systems that can create fresh values and mention real times are of great interest to protocol security. For instance, many distance authentication protocols [15, 2] rely on timing measures. Thus extending our model with real times and determining decidable fragments, *e.g.*, balanced systems, is of great interest for the verification of such protocols. We are also currently implementing these protocols in Maude.

References

1. R. M. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.*, 290(1):695–740, 2003.
2. S. Brands and D. Chaum. Distance-bounding protocols. In EUROCRYPT 1994.
3. A. Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.
4. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude: A High-Performance Logical Framework*. Springer, 2007.
5. S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, 1971.
6. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
7. N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.
8. J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.
9. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. On protection in operating systems. In *SOSP*, 1975.
10. M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. *Inf. Comput.* Accepted for Publication.
11. M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Progressing collaborative systems. In *FCS-PrivMod*, 2010.
12. M. Kanovich, P. Rowe, and A. Scedrov. Policy compliance in collaborative systems. In *CSF*, 2009.
13. M. I. Kanovich, T. B. Kirigin, V. Nigam, A. Scedrov, C. L. Talcott, and R. Perovic. A rewriting framework for activities subject to regulations. In *RTA*, 2012.
14. M. I. Kanovich, P. Rowe, and A. Scedrov. Collaborative planning with confidentiality. *J. Autom. Reasoning*, 46(3-4):389–421, 2011.
15. C. Meadows, R. Poovendran, D. Pavlovic, L. Chang, and P. F. Syverson. Distance bounding protocols: Authentication logic analysis and collusion attacks. In *Advances in Information Security*, 2007.
16. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.
17. V. Nigam, T. B. Kirigin, A. Scedrov, C. L. Talcott, M. I. Kanovich, and R. Perovic. Towards an automated assistant for clinical investigations. In *IHI*, 2012.
18. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theor. Comput. Sci.*, 299(1-3):451–475, 2003.

A Encoding of the Protocol described in Section 3 as a Bounded Memory Protocol

The initial set of facts contains the facts $Guy(A,k)$, $Guy(B,k)$, $Secret(s)$, $Failure(f)$ and $Init(\langle enc(k,\langle t_0,\$,0,t_1\rangle), enc(k,\langle t_1,\langle q,B\rangle,0,t_2\rangle), enc(k,\langle t_2,B,1,t_3\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle)$ specifying that the agents A and B share the uncompromised key k and that $\mathcal{M}$'s initial configuration is encoded in fact $Init$.

The protocol is formalized by the following bounded memory theories for the participants A and B:

Theory for A:
ROLA: $Guy(G,k)Init(I)Secret(s)Failure(f)P(*)$
$\quad\rightarrow Guy(G,k)Init(I)Secret(s)Failure(f)A_0(I,k,s,f)$
UPDA: $A_0(X,k,s,f)P(*) \rightarrow \exists \boldsymbol{t'_i}.A_1(k,s,f)N_S(X')$
RESA0: $A_1(k,s,f)N_R(Y_0) \rightarrow A_2(k,s,f)N_S(s)$
RESA1: $A_1(k,s,f)N_R(Y_1) \rightarrow A_2(k,s,f)N_S(s)$
RESA2: $A_1(k,s,f)N_R(Y_2) \rightarrow A_2(k,s,f)N_S(s)$
RESA3: $A_1(k,s,f)N_R(Y_3) \rightarrow A_2(k,s,f)N_S(f)$
ERASEA: $A_2(k,s,f) \rightarrow P(*)$

where $X = \langle enc(k,\langle t_0,\$,0,t_1\rangle), enc(k,\langle t_1,\langle q,B\rangle,0,t_2\rangle), enc(k,\langle t_2,B,1,t_3\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$,
$X' = \langle enc(k,\langle t'_0,\$,0,t'_1\rangle), enc(k,\langle t'_1,\langle q,B\rangle,0,t'_2\rangle), enc(k,\langle t'_2,B,1,t'_3\rangle), enc(k,\langle t'_4,B,1,t'_5\rangle)\rangle$,
$Y_0 = \langle enc(k,\langle t_{00},\langle q_0,\xi\rangle,0,t_{01}\rangle), enc(k,\langle t_{01},\alpha_1,0,t_{20}\rangle), enc(k,\langle t_{20},\alpha_2,e_2,t_{21}\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$,
$Y_1 = \langle enc(k,\langle t_{00},\alpha_0,0,t_{01}\rangle), enc(k,\langle t_{01},\langle q_0,\xi\rangle,0,t_{20}\rangle), enc(k,\langle t_{20},\alpha_2,e_2,t_{21}\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$,
$Y_2 = \langle enc(k,\langle t_{00},\alpha_0,0,t_{01}\rangle), enc(k,\langle t_{01},\alpha_1,0,t_{20}\rangle), enc(k,\langle t_{20},\langle q_0,\xi\rangle,e_2,t_{21}\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$,
$Y_3 = \langle enc(k,\langle t_{00},\alpha_0,0,t_{01}\rangle), enc(k,\langle t_{10},\alpha_1,0,t_{12}\rangle), enc(k,\langle t_{20},\alpha_2,e_2,t_{21}\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$.

Theory for B:
ROLB: $Guy(G,k)P(*) \rightarrow Guy(G,k)B_0(k)$
UPDB: $B_0(k)N_R(X) \rightarrow \exists \boldsymbol{t'_i}.B_1(k)N_S(X')$
MOVEB: $B_0(k)N_R(Y) \rightarrow \exists \boldsymbol{t'_i}B_1(k)N_S(Y')$
ERASEB: $B_1(k) \rightarrow P(*)$

where $X = \langle enc(k,\langle t_0,\xi_0,0,t_1\rangle), enc(k,\langle t_1,\langle q,\xi\rangle,0,t_2\rangle), enc(k,\langle t_2,\xi_2,1,t_3\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$,
$X' = \langle enc(k,\langle t'_0,\xi_0,0,t'_1\rangle), enc(k,\langle t'_1,\langle q,\xi\rangle,0,t'_2\rangle), enc(k,\langle t'_2,\xi_2,0,t'_3\rangle), enc(k,\langle t'_3,B,1,t'_4\rangle)\rangle$,
$Y = \langle enc(k,\langle t_0,\xi_0,0,t_1\rangle), enc(k,\langle t_1,\langle q,\xi\rangle,0,t_2\rangle), enc(k,\langle t_2,\xi_2,e_2,t_3\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$
where $e_2 \neq 1$,
$Y' = \langle enc(k,\langle t_0,\xi_0,0,t'_1\rangle), enc(k,\langle t'_1,\eta,0,t'_2\rangle), enc(k,\langle t'_2,\langle q',\xi_2\rangle,0,t_3\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$
if $q\xi \rightarrow q'\eta R$ is an $\mathcal{M}$'s instruction or
$Y' = \langle enc(k,\langle t_0,\langle q',\xi_0\rangle,0,t'_1\rangle), enc(k,\langle t'_1,\eta,0,t'_2\rangle), enc(k,\langle t'_2,\eta_2,0,t_3\rangle), enc(k,\langle t_4,B,1,t_5\rangle)\rangle$
if $q\xi \rightarrow q'\eta L$ is an $\mathcal{M}$'s instruction.

There is a MOVEB rule for each instruction of $\mathcal{M}$. Hence, the reduction is polynomial on the number of instructions in $\mathcal{M}$. Both theories for A and for B have the corresponding role generation rules ROLA and ROLB, which create new sessions, as well as rules ERASEA and ERASEB, which delete role state predicates of completed sessions. As previously discussed,this allows traces to have an unbounded number of protocol sessions.