Bounded memory Dolev-Yao adversaries in collaborative systems

Max Kanovich

Queen Mary, University of London, UK

Tajana Ban Kirigin

University of Rijeka, HR

Vivek Nigam

Universidade Federal da Paraíba, João Pessoa, Brazil

Andre Scedrov

University of Pennsylvania, Philadelphia, USA

Abstract

This paper extends existing models for collaborative systems. We investigate how much damage can be done by insiders alone, without collusion with an outside adversary. In contrast to traditional intruder models, such as in protocol security, all the players inside our system, including potential adversaries, have similar capabilities. They have bounded storage capacity, that is, they can only remember at any moment a bounded number of symbols. This is technically imposed by only allowing balanced actions, that is, actions that have the same number of facts in their pre- and post-conditions, and bounding the size of facts, that is, the number of symbols they contain. On the other hand, the adversaries inside our system have many capabilities of the standard Dolev-Yao intruder, namely, they are able, within their bounded storage capacity, to compose, decompose, overhear, and intercept messages as well as create fresh values. We investigate the complexity of

Email addresses: mik@eecs.qmul.ac.uk (Max Kanovich), bank@math.uniri.hr (Tajana Ban Kirigin), vivek.nigam@gmail.com (Vivek Nigam), scedrov@math.upenn.edu (Andre Scedrov)

the decision problem of whether or not an adversary is able to discover secret data. We show that this problem is PSPACE-complete when the size of messages is an input bound and when all actions are balanced and can possibly create fresh values. As an application, we turn to security protocol analysis and demonstrate that many protocol anomalies, such as the Lowe anomaly in the Needham-Schroeder public key exchange protocol, can also occur when the intruder is one of the insiders with bounded memory.

Keywords: Collaborative Systems, Protocol Security, Complexity Results

---

1. Introduction

This paper extends existing models for collaborative systems [31, 32] by allowing creation of *fresh values*, often called *nonces* in protocol security literature. Fresh values are essential not only in protocol security but also in other administrative processes that require for example unique identification. For instance, when a bank customer opens a new bank account, a unique bank number account is assigned to it. As each bank account is assigned a fresh number, one can uniquely identify accounts. Similar uses of fresh values also appear in other collaborative systems, for example in clinical trials. In particular, each subject participating in a clinical trial is assigned a unique identification code [9]. Such identification code is then used to, among other things, keep track and maintain the subject's trial record.

In collaborative systems agents collaborate to reach some common goal. Since they do not trust each other completely, they do not want some sensitive information, such as a password, to leak to some other agent. Therefore one is interested in showing that the system is secure, *i.e.*, such critical states cannot be reached. In [32], Kanovich *et al.* proposed a model for collaborative systems based on *multiset rewriting*. Later in [31], Kanovich *et al.* proposed three compliance problems for collaborative systems called *weak plan compliance*, *plan compliance*, and *system compliance*. This paper extends this model by allowing multiset rewrite rules or actions to create fresh values and investigates the complexity of these three problems for our extended model.

Following [32], we assume here that all actions in our system are *balanced*, that is, they have the same number of facts in their pre- and post-conditions. If we additionally bound the size of facts, *i.e.*, the total number of function and constant symbols a fact can contain, then all participants or agents inside our system have

a bounded storage capacity. That is, at any moment, they can only remember a bounded number of symbols.

We show that all three problems mentioned above are PSPACE-complete when the size of facts is an input bound and when systems contain only balanced actions that can possibly create fresh values. PSPACE membership is not trivial. As plans may contain exponentially many actions and each action may create a fresh value, plans may contain an exponential number of fresh constants, which in principle precludes PSPACE membership. We cope with this problem by reusing obsolete constants instead of creating new constant names.

Although our initial efforts were in collaborative systems, we realized that our results also have important consequences for the domain of protocol security. We model an intruder that is a malicious agent inside the system and investigate how much damage can be done by insiders alone, without collusion with an outside adversary. Since all players inside our system have bounded storage capacity, so do the adversaries. This contrasts with traditional intruder models, which normally include a powerful Dolev-Yao intruder [18] that has an unbounded memory. On the other hand, our bounded memory adversaries and the standard Dolev-Yao intruder [18] share many capabilities, namely, they are able, within their bounded storage capacity, to compose, decompose, overhear, and intercept messages as well as create fresh values.

We show that the secrecy problem of whether or not an adversary can discover a secret is PSPACE-complete when the size of messages is an input bound and when actions are balanced and can create fresh values. This contrasts with previous results in protocol security literature [19], where it has been shown that the same problem is undecidable even when the size of messages is fixed. However, there the intruder was allowed to have unbalanced actions, and consequently intruder's memory was not necessarily bounded.

We further demonstrate that when our adversary has *enough* storage capacity, then many protocol anomalies, such as the Lowe anomaly [35] in the Needham-Schroeder public key exchange protocol [39], can also occur in the presence of a bounded memory intruder. We believe that this is one reason for the successful use of model checkers in protocol verification in the past years. Moreover, we also provide *some quantitative measures* for the security of protocols, namely, the smallest amount of memory needed by the intruder to carry out anomalies for a number of protocols, such as Needham-Schroeder [39, 35], Yahalom [14], Otway-Reese [14, 48], Woo-Lam [14], and Kerberos 5 [8, 10].

In the first part of this paper, we introduce the complexity results obtained and in the second part of the paper, we demonstrate how our theoretical results can be

applied to protocol security. We now summarize our main contributions. After introducing the main vocabulary and the decision problems in Section 2:

- We show that plans constructed using balanced actions can be exponentially long (Section 3), thus precluding PSPACE membership;
- We show that when the size of facts is bounded and systems have only balanced actions that can create fresh values, it is enough to fix a priori a set with a few fresh names. The idea is that instead of creating new names, one reuses names from the fixed set of fresh values (Section 4);
- We prove the complexity results for the decision problems introduced in Section 2 when bounding the size of facts and using balanced systems that can create fresh values (Section 5);

After investigating the complexity of the decision problems introduced in Section 2, we apply our results in the domain of protocol security.

- We introduce a balanced protocol theory and a balanced intruder theory (Section 6). Then we demonstrate that many protocol anomalies can also be carried out by a bounded memory intruder, namely, Needham-Schroeder [39, 35], Yahalom [14], Otway-Reese [14, 48], Woo-Lam [14], and Kerberos 5 [8, 10]. The detailed encoding of the Lowe anomaly for the Needham-Schroeder protocol is shown in Section 6.3. The remaining anomalies can be found in the technical report [26].
- We prove the complexity results for the secrecy problem when bounding the size of messages and when using balanced systems specifying protocol theories with a bounded memory intruder (Section 7).

Finally, we discuss related work and conclude by pointing out some future work in Sections 8 and 9.

This paper is an extended and improved version of [27].

2. Preliminaries

In this section we review the main vocabulary and concepts introduced in [31, 32] and extend their definitions to accommodate actions that can create fresh values and introduce an adversary.

2.1. Multiset Rewriting

At the lowest level, we have a first-order alphabet Σ (also called signature in formal verification papers) that consists of a set of sorts (or types) together with the predicate symbols $P_1, P_2, \ldots$, function symbols $f_1, f_2, \ldots$, and constant symbols $c_1, c_2, \ldots$ all with specific sorts. The multi-sorted terms over the alphabet are expressions formed by applying functions to arguments of the correct sort. Since terms may contain variables, all variables must have associated sorts. A *fact* is an atomic predicate over multi-sorted terms. Facts have the form $P(t_1, \ldots, t_n)$ where P is an n-ary predicate symbol and $t_1, \ldots, t_n$ are terms, each with its own sort.

Definition 2.1. The *size of a fact* is the total number of term and predicate symbols it contains. We count one for each predicate and function name, and one for each constant and variable symbol. We use $|F|$ to denote the size of a fact F.

For example, $|P(b, c)| = 3$ and $|P(f(b, n, b), z)| = 6$. In this paper, as in [19], we will normally assume an upper bound on the size of facts.

A *state*, or *configuration* of the system is a finite multiset W of *grounded* facts, *i.e.*, facts that do not contain variables. We use both WV and W, V to denote the multiset resulting from the multiset union of W and V. A multiset rewriting system (MSR) is a set of multiset rewrite rules, which are used to change configurations. Rules have the form $W \rightarrow W'$. All free variables appearing in the rule are assumed to be universally quantified. By applying a rule for a ground substitution (σ), the multiset W under this substitution ($W\sigma$) is replaced with the multiset W' under the same substitution ($W'\sigma$). Hence, this rule can be applied to the configuration $V, W\sigma$, called *enabling configuration*, to obtain the configuration $V, W'\sigma$.

Definition 2.2. The size of a configuration $\mathcal{S}$ is the total number of facts in $\mathcal{S}$.

Intuitively, a configuration specifies a snapshot of the state of the world, while rules specify operations that change the state of the world. One is often interested in determining whether in a multiset rewrite system some configuration is reachable from another configuration. This problem is called the *reachability problem*. Formally, given a set $\mathcal{R}$ of multiset rewrite rules, if there is a sequence of (0 or more) rules from $\mathcal{R}$ which transforms W into Z, then we say that Z is *reachable* from W using $\mathcal{R}$.

Rules that can create fresh values. The rewrite rules of the form above have an important limitation, namely, one cannot specify the creation of *fresh values*. These values are normally called *nonces* in protocol security literature. Fresh values are often used in administrative processes. For example, when one opens a new bank account, the number assigned to the account has to be fresh, that is, it has to be different from all the other existing bank account numbers. Similarly, whenever a bank transaction is initiated, a fresh number is assigned to the transaction, so that it can be uniquely identified. Fresh values are also used in the execution of protocols. At some moment in a protocol run, an agent might need to create a *nonce* that is not known to any other agent in the network. This nonce, when encrypted in a message, is then usually used to establish a secure communication among agents.

As in [19], we borrow the same notion of freshness from proof theory to specify rules that can create fresh values. In particular, the elimination rule in natural deduction systems [22, 42] for the existential quantifier introduces a fresh value, also called *eigenvariable*. This rule is often written in the following way

$$\frac{\exists x.\phi \qquad \begin{array}{c}\phi[c/x]\\ \vdots \\ \psi\end{array}}{\psi}\ \exists_E$$

with the side condition that *the constant c does not appear in any other hypothesis*. The rule above states that if we have a proof of the formula $\exists x.\phi$ and if we have a proof of ψ using the hypothesis $\phi[c/x]$ then we have a proof of ψ. The side condition means that the only hypothesis in the proof of ψ that contains c is $\phi[c/x]$. That is, the constant c is a fresh constant introduced in the premises of the elimination rule.

Following the notion of freshness above, we allow rewrite rules to create fresh values. These rules have the form $W \rightarrow \exists \vec{z}.W'$ and specify that the existentially quantified variables, $\vec{z}$, are to be replaced with fresh values, that is, with values that do not appear in the enabling configuration nor in the ground terms replacing the free variables in the rule. For example, we can apply the rule $P(x)\ Q(y) \rightarrow \exists z.R(x,z)\ Q(y)$ to the configuration $V\ P(t)\ Q(s)$ to get the configuration $V\ R(t,c)\ Q(s)$, where the constant c must be fresh.

As we will illustrate later, rules that can create fresh values play an important role in the specification of collaborative systems and security protocols. For example, whenever a bank transaction is initiated, one can specify that a fresh

number is to be assigned to the transaction by using a rule of the form:

$$Transaction(\text{noID}, user) \rightarrow \exists id.Transaction(id, user)$$

where noId is a constant denoting that a transaction has no identification number. When this rule is applied, its semantics ensures that the value replacing variable id is fresh. Therefore, each transaction can be uniquely identified using the transactions identification number created.

Finally, we would also like to point out that [11, 28] provides a precise connection between the operational semantics of MSRs containing rules that can possibly create fresh values and linear logic derivations [23].

Applications of MSRs. Multiset rewriting systems have been used in several domains. For instance, it has been shown that a wide range of algorithms [5], Artificial Intelligence problems [33, 32], security protocols [19] as well collaborative systems [32, 28] can be specified by MSRs. In Section 3, we show a MSR specification of the well-known Towers of Hanoi puzzle and in Section 6 we show how protocol theories can be specified by using MSRs.

2.2. Local State Transition Systems

In a collaborative system, agents collaborate to achieve a common goal, but they do not trust each other completely. Therefore, while collaborating, an agent might be willing to share some of his private information with some agents, but not willing to share some other information. As an example, a patient would share his medical history with a doctor, but would not disclose to the doctor some other information such as his bank account PIN number.

In order to specify private and shared information, [31, 32] introduced Local State Transition Systems (LSTS). In LSTSes the global configuration is partitioned into different local configurations each of which is accessible only to one agent. There is also a public configuration, which is accessible to all agents. Intuitively, local configurations contain the data that are private to the agents of the system, while the global configuration contains the data that are public, *i.e.* accessible to all agents. This separation of the global configuration is done by partitioning the set of predicate symbols in the alphabet and it will be usually clear from the context. Predicate symbols are typically annotated with the name of the agent that owns it or with pub if it is public. For instance, the fact $P_A(\vec{t})$ belongs to the agent A, while the fact $P_{pub}(\vec{t})$ is public. This paper adopts the same approach to specify private and shared information. However, to formally specify the secrecy problem later in this Section, we also assume that among the agents

in the system, there is an adversary M. Moreover, we also assume the existence of a special constant s in the alphabet Σ denoting the secret that should not be discovered by the adversary.

As in [31, 32], each agent has a finite set of *actions* or *rules*, which transform the global configuration. Here, as in [19, 28, 11], we allow agents to have more general actions that can create fresh values. Following the intuition above, an agent can only have access to his own local configuration, containing his private data, and the public configuration, containing data that are available to all agents. This is formalized by restricting the facts that can be mentioned in a rule. In particular, actions that belong to an agent A have the form:

$$W_A\, W_{pub} \rightarrow_A \exists \vec{z}. W'_A\, W'_{pub}. \tag{1}$$

The multisets W_A and W'_A contain only facts belonging to the agent A and the multisets W_{pub} and W'_{pub} contain only public facts. The multiset $W_A\ W_{pub}$ is the pre-condition of the action and the multiset $W'_A\ W'_{pub}$ is the post-condition of the action. Actions work as multiset rewrite rules, where all free variables in a rule are treated as universally quantified.

The main novelty of this paper in comparison with [31, 32] is that we allow rules to create fresh values, specified by the existentially quantified variables $\vec{z}$ appearing in the rule. As in MSRs, they denote that the variables $\vec{z}$ appearing in the post-condition have to be replaced with *fresh values*.

Rules of the above form impose the restriction that any fresh value created by an agent A appears only in facts belonging to A and/or in public facts. Since an agent does not have access to the facts belonging to the other agents, if he wants to share some fresh value, then he needs to publish it in the public configuration. This can be done by applying a single instance of an action, such as the one below:

$$Q_A(x)\ R_{pub}(x) \rightarrow_A \exists z. Q_A(z)\ R_{pub}(z)$$

where the values in the private and public facts Q_A and R_{pub} are updated by a fresh value. If an agent does not want to share a fresh value, but only store the fresh value in his local configuration, then this can also be specified by using existentially quantified variables only in private facts. This is illustrated by the following action, which does not contain public facts:

$$Q_A(x) \rightarrow_A \exists z. Q_A(z)$$

Since the variable z does not appear in any public fact, the fresh value created is not shared to the public. Finally, agents can learn fresh values that have been

shared by copying them into private facts, *e.g.*by applying the following action:

$$R_{pub}(x) \rightarrow_A Q_A(x)\ R_{pub}(x)\ .$$

When this action is applied, the agent A learns x as it is copied to his own local configuration.

For simplicity, we often omit the name of agents from actions and predicates when the agent is clear from the context.

Definition 2.3. A *local state transition system* (LSTS) T is a tuple $\langle \Sigma, I, M, R_T, s \rangle$, where Σ is the alphabet of the language, I is a set of agents, $M \in I$ is the adversary, R_T is the set of actions owned by the agents in I, and s is the secret.

We use the notation $W \rhd_T U$ or $W \rhd_r U$ to mean that there is an action r in the LSTS T which can be applied to the configuration W to transform it into the configuration U. We let $\rhd_T^+$ and $\rhd_T^*$ denote the transitive closure and the reflexive, transitive closure of $\rhd_T$ respectively. Usually, however, agents do not care about the entire configuration of the system. Instead, they are interesed in whether a configuration contains some particular facts. Therefore, we use the notion of partial goals. We write $W \rightsquigarrow_T Z$ or $W \rightsquigarrow_r Z$ to mean that $W \rhd_r ZU$ for some multiset of facts U. For example with the action $r : X \rightarrow_A Y$, we find that $WX \rightsquigarrow_r Y$, since $WX \rhd_r WY$. We define $\rightsquigarrow_T^+$ and $\rightsquigarrow_T^*$ to be the transitive closure and the reflexive, transitive closure of $\rightsquigarrow_T$ respectively. We say that the partial configuration Z is reachable from configuration W using T if $W \rightsquigarrow_T^* Z$. We also consider configurations that are reachable using the actions from all agents except for one. Thus we write $X \rhd_{-A_i}^* Y$ to indicate that Y can be reached exactly from X without using the actions of agent A_i. Finally, given an initial configuration W and a partial configuration Z, we call a *plan* any sequence of actions that leads from configuration W to a configuration containing Z.

Example. As an illustrative example, consider the scenario adapted from [32] where a patient needs a medical test, *e.g.*, a blood test, to be performed in order for a doctor to correctly diagnose the patient's health. This process may involve several agents, such as a patient, a nurse, and a lab technician. Each of these agents has his own set of tasks. For instance, the patient's initial task could be to make an appointment and go to the hospital. Then, the secretary would send the patient to the nurse who would collect the patient's blood sample and send it to the lab technician, who would finally perform the required test. This scenario can

be specified as an LSTS. The following rules specify some of the actions of the agents N (nurse) and L (lab technician) from this scenario:

$$
\begin{array}{lll}
Nurse(\text{blank}, \text{blank}, \text{blank})\ Pat(name, test) & & \\
 & \rightarrow_N & Nurse(name, \text{blank}, test)\ Pat(name, test) \\
Nurse(x, \text{blank}, \text{blood}) & \rightarrow_N & \exists id.Nurse(x, id, \text{blood}) \\
Nurse(x, id, \text{blood}) & \rightarrow_N & Lab(id, \text{blood})\ Nurse(x, id, \text{blood}) \\
Lab(id, \text{blood}) & \rightarrow_L & TestResult(id, \text{result})
\end{array}
$$

The predicates Pat, Lab and $TestResult$ are public, while the predicate $Nurse$ belongs to the nurse. Here "blank" is the constant denoting an unknown value, "blood" is the constant denoting the type of test that is a blood test, "result" is one of the constants from the set denoting the possible test outcomes, while $test$, $name$, x and id are all variables. The most interesting action is the second action, which generates a fresh value. This fresh value is an identification number assigned to the test required by the patient. Then in the third action, when the nurse sends a request the lab technician to perform a blood test, the nurse does not provide the name of the patient, but instead, in order to anonymize the patient, she only sends the identification number generated. Finally, in the last action, the lab technician makes available the test results attached with the corresponding identification number. In order not to mix up the test result of one patient with test result of another patient, each patient (sample) should have a different identification number assigned. In the specification above, this is enforced by the second rule since a fresh value is created.

In this particular example, there is no secret involved. However, there are undesirable situations that have to be avoided. In particular, the test results of a patient should not be publicly leaked with the patient's name. These situations will be specified by using *critical configurations* introduced later in this section.

Nonces are important in protocol security and in many other administrative processes that require unique identification and security, such as contract signing protocols [9]. Recently we have been applying collaborative systems in the domain of medical research and drug development, to model clinical investigations (CIs) that test a new drug on human subjects [41]. CIs are rigorously regulated by policies elaborated by governmental agencies such as the Food and Drug Administration (FDA). Other regulations may be imposed by multiple institutional policies and protocols. Due to the complexity of both regulations and activities, there is great potential for violation due to human error, misunderstanding, or even intent.

Fresh values play an important role in the specification of clinical trials. Due to confidentiality and privacy concerns that are clearly involved here, nonces are essential in modeling for example identification codes assigned to every subject participating in a CI. For instance, the following paragraph in Section 4.1.2.3 of the implementation manual of Study Data Tabulation Model [9], a standard on how data should be collected and stored during a trial, mentions the use of unique identification codes:

> "To identify a subject uniquely across all studies for all applications or submissions involving the product, a unique identifier (USUBJID) should be assigned and included in all datasets. [$\cdots$] This means that no two (or more) subjects, across all trials in the submission, may have the same USUBJID."

In particular, each subject should have a unique identification code USUBJID. As illustrated above, the assignment of these codes can be modeled by using actions that create fresh values.

Moreover, some CIs may have blind trials, where subjects do not know whether they are taking the actual drug being tested or a placebo. In such trials, the drug and the placebo themselves have to be delivered to locations properly boxed and labelled, *e.g.*, lot number LNR1305, box no. 11 together with expiry date. These labels can also be modeled by nonces, so that the boxes can be appropriately assigned to specific subjects. Additionally, if some problems are detected such as temperature excursion of the drug in transport, sun exposure, etc, the drug can be uniquely identified and retrieved from all locations as per lot number.

Balanced Actions. A central assumption in this paper is that of balanced actions. We classify an action as *balanced* if the number of facts in its pre-condition is the same as the number of facts in its post-condition. As discussed in [32], balanced actions have the special property that when applied they preserve the size of configurations, *i.e.*, the number of facts in configurations. This is because when applying a balanced action the same number of facts deleted from an enabling configuration is then inserted into the resulting configuration. Hence, if an LSTS has only balanced actions, then all configurations in a plan have the same number of facts. In particular, the size of all configurations in a plan is the same as the size of the initial configuration.

On the one hand, when using unbalanced actions it is possible to create a fact without consuming a fact in the process. For example, the following action

creates a fact: $\rightarrow_A Q_A(x)$. By using this action, one could for instance expand a configuration by creating new facts an unbounded number of times. Hence, the size of configurations appearing in a plan obtained using unbalanced actions may be unbounded. This seems to be a cause for the undecidability of many problems that we consider in this paper, such as the secrecy problem. On the other hand, creating a new fact using a balanced action amounts to inserting that fact into the resulting configuration by replacing a fact appearing in the enabling configuration. In order to support the creation of new facts in balanced systems, we use *empty facts*, $P(*)$. An empty fact intuitively denotes a slot available that could be filled by a new fact. For instance, the following balanced action creates a non-empty fact by consuming an empty fact:

$$P(*) \rightarrow_A Q_A(x).$$

This action specifies that a free slot can be filled by the fact $Q_A(x)$. Symmetrically, an agent can replace a non-empty with an empty fact, as specified by the following rule:

$$Q_A(x) \rightarrow_A P(*).$$

Intuitively, this rule specifies that the fact $Q_A(x)$ is forgotten freeing up memory. The empty fact created by this rule could then be reused by another rule that requires an empty fact.

By using empty facts, $P(*)$, one can also transform unbalanced systems into balanced systems. For instance, in the medical example shown above, all actions are balanced, except the action:

$$Nurse(x, id, \text{blood}) \rightarrow_N \ Lab(id, \text{blood})\ Nurse(x, id, \text{blood}).$$

In particular, its pre-condition has less facts than its post-condition. We can modify this action so that it is transformed into a balanced action by adding an empty fact to its pre-condition, thereby obtaining the following balanced action:

$$P(*)\ Nurse(x, id, \text{blood}) \rightarrow_N \ Lab(id, \text{blood})\ Nurse(x, id, \text{blood}).$$

In order for the Nurse to ask the lab for more tests, she needs to check whether there is an empty fact available. One could interpret this as the nurse checking whether the lab has enough capacity to perform another test. Otherwise, the nurse will have to wait until a $P(*)$ is made available. This could happen, for instance, when some patient receives his test results and therefore no longer requires the

test to be carried out, which can be specified by the following rule:

$$Nurse(name, id, blood),\ TestResult(id, \text{result}),\ Pat(name, blood) \\ \rightarrow_N Nurse(name, id, blood)\ Rec(name, \text{result})\ P(*)$$

Once the test result of a patient is available and delivered to the patient, the Nurse can use the $P(*)$ fact created to request a new test for another patient to be carried by the lab technician. Notice that the test results are still stored in the patient's medical records, specified by the private fact Rec belonging to the Nurse.

As illustrated above, the use of balanced actions bounds the number of facts agents can remember, but this condition alone does not bound the memory of an agent, that is, the number of symbols he can remember. To bound the memory of the agents of a system, one needs to additionally assume that facts have a bounded size. That is, there is a maximum number of symbols a fact can contain. Otherwise, if we do not impose a bound to the size of facts, agents could for instance use a pairing function, $\langle \cdot, \cdot \rangle$, and facts with unbounded depth to remember as many constants (or data) they need. For example, instead of using n facts, $Q(c_1), \ldots, Q(c_n)$, to store n constants, $c_1, \ldots, c_n$ for some n, an agent could store all of these constants by using the single fact $Q(\langle c_1, \langle c_2, \langle \cdots, \langle c_{n-1}, c_n \rangle \rangle \cdots \rangle \rangle)$.

Intuitively, by using balanced systems and assuming such a bound on the size of facts, we obtain a bound on the number of slots available for predicate, function, and constant symbols in any configuration of a run. As we will discuss in Section 4, this bound will be key to obtain the decidability of the decision problems that we investigate in this paper, such as the secrecy problem.

Notice as well that such an upper bound on the size of facts was also assumed in previous work [19], while [32, 31] assumed a fixed bound on the size of facts.

Critical Configurations. In order to achieve a final goal, it is often necessary for an agent to share some private knowledge with another agent. However, although an agent might be willing to share some private information with some agents, he might not be willing to do the same with other agents. For example, a patient might be willing to share his test results with the nurse, but not with the lab technician. In such scenarios, one is interested in determining if a system complies with some *confidentiality policies*, such as a patient's test result should not be publicly available together with his name.

Formally, a confidentiality policy of an agent is a set of partial configurations that this agent considers undesirable or bad. A configuration is called *critical for an agent* if it contains one of the partial configurations from his policy, and it is

simply called *critical* if it is critical for some agent of the system. We classify any plan that does not reach any critical configuration as *compliant*.

In this paper, we make an additional assumption that critical configurations are closed under renaming of nonce names, that is, if W is a critical configuration and $W\sigma = W'$ where σ is substitution renaming the nonces in W, then W' is also critical. This is a reasonable assumption since critical configurations are normally defined without taking into account the names of nonces used in a particular plan, but only how they relate in a configuration to the initial set of symbols in Σ and amongst themselves. For instance, in the medical example above consider the following configuration $\{TestResult(n_1, result), Tec(n_1, paul)\}$, where Tec is a predicate belonging to the lab technician. This configuration is critical because the lab technician knows Paul's test results, $result$, since she knows his identity number, denoted by the nonce n_1, and the name that is associated to this identifier. Using the same reasoning, one can easily check that the configuration resulting from renaming the nonce n_1 is also critical. In [34] it is pointed out that in the scenarios involving the privacy of medical data what matters are the categories of participants (*e.g.*, physicians, nurses, or patients) other then the actual individuals in these categories.

Definition of Problems. We review the three policy compliances introduced in [31, 32] and the secrecy problem related to protocol security. This paper makes the additional assumption that initial, goal and critical configurations are closed under renaming of nonces.

- (System compliance) Given a local state transition system T, an initial configuration W, a (partial) goal configuration Z, and a set of critical configurations, is no critical state reachable from W, and does there exist a plan leading from W to Z?

- (Weak plan compliance) Given a local state transition system T, an initial configuration W, a (partial) goal configuration Z, and a set of critical configurations, is there a compliant plan that leads from W to Z?

- (Plan compliance) Given a local state transition system T, an initial configuration W, a (partial) goal configuration Z, and a set of critical configurations, is there a compliant plan that leads from W to Z such that for each agent A_i and for each configuration Y along the plan, whenever $Y \rhd^*_{-A_i} V$, then V is not critical for A_i?

- (Secrecy problem) Given a local state transition system T, is there a plan from the given initial configuration to a configuration in which the adversary M owns the fact $M(s)$,[1] where s is a secret originally owned by another participant?

Intuitively, a system is system compliant if whatever actions the agents perform, no undesired state for any agent is reached and if there is a compliant plan, where the agents reach a common goal. On the other hand, a weak plan compliant system is a system that has a compliant plan. However, if some agent of the system does not follow the compliant plan, then it can happen that an undesired state for some agent is reached. Finally, a plan compliant system is such that there is a compliant plan and moreover if an agent A_i wants to stop collaborating, then it is guaranteed that the remaining agents are not able to reach any of A_i's undesired states.

The type of compliance, *i.e.*, weak plan, system, or plan compliance, that is considered appropriate will depend on the type of collaborative system in question. In some cases, such as in the medical scenario above, one might require system compliance: according to hospital policies, it should never be possible that, for example, the lab technician gets to know the test results of the patient. In other cases, however, such as when researchers are collaborating to write a paper before a deadline, weak plan compliance might be more appropriate. The collaborating researchers are only interested to know whether there is a compliant plan where the goal of writing the paper before the deadline is achieved. They trust each other to be well intentioned in contributing to the paper being written on time. Further examples illustrating and motivating the definitions above can be found in [32, 31].

The secrecy problem is basically an instantiation of the weak plan compliance problem with no critical configurations. It is interesting to note that this problem can also be seen as a kind of a dual to the weak plan compliance problem; is there is a plan from the initial configuration to a *critical configuration* where the adversary M owns the secret s, originally owned by another participant? What we mean by owning a secret s, or any constant c in general, is that the agent has a private fact $Q(c')$ such that c is a subterm of c'.

[1] M is a predicate name belonging to the intruder.

3. Examples of exponentially long plans

In this section, we illustrate that plans can, in principle, be exponentially long. In particular, we discuss an encoding of the well-known puzzle of the Towers of Hanoi. Such plans seem to preclude PSPACE membership, especially when nonces are involved, since there can be *a priori* an exponential number of nonces in such plans. We will later show, in Section 4, how to circumvent this problem by reusing obsolete constants instead of creating new names for fresh values.

3.1. Towers of Hanoi

Towers of Hanoi is a well-known mathematical puzzle. It consists of three pegs b_1, b_2, b_3 and a number of disks a_1, a_2, a_3, ... of different sizes, which can slide onto any peg. The puzzle starts with the disks neatly stacked in ascending order of size on one peg, the smallest disk at the top. The objective is to move the entire stack stacked on one peg to another peg, obeying the following rules:

(a) Only one disk may be moved at a time.

(b) Each move consists of taking the upper disk from one of the pegs and sliding it onto another peg, possibly on top of the other disks that may already be present on that peg.

(c) No disk may be placed on top of a smaller disk.

The puzzle can be played with any number of disks and it is known that the minimal number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$, where n is the number of disks.

The problem can be represented by an LSTS. We introduce the type $disk$ for the disks, type $diskp$ for either disks or pegs, with $disk$ being a subtype of $diskp$. The constants a_1, a_2, a_3, ..., a_n are of type $disk$ and b_1, b_2, b_3 of type $diskp$. We use facts of the form $On(x, y)$, where x is of type $disk$ and y is of type $diskp$, to denote that the disk x is either on top of the disk or on the peg y. Facts of the form $Clear(x)$, where x is of type $diskp$, denote that the top of the disk x is clear, *i.e.*, no disk is on the top of or on x, or that no disk is on the peg x. Since disks need to be placed according to their size, we also use facts of the form $S(x, y)$, where x is of type $disk$ and y is of type $diskp$, to denote that the disk x can be put on top of y. In our encoding, we make sure that one is only allowed to put a disk on top of a larger disk or on an empty peg, *i.e.*, that x is smaller than y in the case of

y being a disk. This is encoded by the following facts in the initial configuration:

$$\begin{array}{rcl} S(a_1,a_2)\ S(a_1,a_3)\ S(a_1,a_4) & \ldots & S(a_1,a_n)\ S(a_1,b_1)\ S(a_1,b_2)\ S(a_1,b_3) \\ S(a_2,a_3)\ S(a_2,a_4) & \ldots & S(a_2,a_n)\ S(a_2,b_1)\ S(a_2,b_2)\ S(a_2,b_3) \\ & \vdots & \\ & S(a_{n-1},a_n)\ S(a_{n-1},a_n)\ S(a_{n-1},b_1)\ S(a_{n-1},b_2)\ S(a_{n-1},b_3) & \end{array}$$

The initial configuration also contains the facts that describe the initial placing of the disks:

$$\begin{array}{c} On(a_1,a_2)\ On(a_2,a_3)\ldots On(a_{n-1},a_n)\ On(a_n,b_1) \\ Clear(a_1)\ \ Clear(b_2)\ \ Clear(b_3)\,, \end{array}$$

The goal configuration consists of the following facts and encodes the state where all the disks are stacked on the peg b_3:

$$\begin{array}{c} On(a_1,a_2)\ On(a_2,a_3)\ldots On(a_{n-1},a_n)\ On(a_n,b_3) \\ Clear(a_1)\ \ Clear(b_1)\ \ Clear(b_2) \end{array}$$

Finally, the only action in our system is:

$$Clear(x)\ On(x,y)\ Clear(z)\ S(x,z) \rightarrow Clear(x)\ Clear(y)\ On(x,z)\ S(x,z)$$

where x has type $disk$, while y and z have type $diskp$. Notice that the action above is balanced. This action specifies that if there is a disk, x, that has no disk on top, it can be either moved to the top of another disk, z, that also has no disk on top, provided that x is smaller than z, specified by predicate $S(x,z)$, or onto a clear peg.

The above Towers of Hanoi puzzle representation with LSTSes can be suitably modified so that each move in this game is identified/accompanied by replacing a previous "ticket" with a fresh ticket.[2] This is accomplished, for example, by the following two rules.

$$\begin{array}{l} T(t)\ Clear(x)\ On(x,y)\ Clear(z)\ S(x,z) \rightarrow \\ \qquad\qquad\qquad\qquad P(*)\ Clear(x)\ Clear(y)\ On(x,z)\ S(x,z) \\ P(*) \rightarrow \exists z.T(z) \end{array}$$

[2]Although the use of tickets is not necessary for solving the Towers of Hanoi problem, it is an illustrative example that in principle one may require an exponential number of fresh values.

The first rule replaces the old ticket $T(t)$ with the empty fact $P(*)$. Then the second rule specifies that a new ticket can be created in exchange of a $P(*)$ fact. If we include a single $P(*)$ fact in the initial configuration above, then it is easy to check that for every move performed in the game, a new fresh value could in principle be created. As before, given n disks, all plans must be of the exponential length $2^n - 1$, at least. Consequently, within the modified version, a plan that creates a different fresh value for every move would contain an exponential number of different fresh values.

However, one does not necessarily need to use an exponential number of different tickets. In fact, since the ticket used in a move is forgotten in the first rule, the same ticket name can be reused as the fresh value in the second rule to enable the next move. Therefore, one can show that there is a plan where the problem is solved with only one ticket.

Although in this particular problem one just needs a single fresh value, for LSTSes in general, more fresh values may be required. We show in the next section that only a few fresh values are needed when we assume a bound on the size of facts and when all actions are balanced.

4. Polynomial Bound for the Number of Fresh Values

As illustrated by the example given in the previous section, plans can be exponentially long and involve an exponential number of fresh values. The use of an exponential number of fresh values seems to preclude PSPACE membership of all the compliance problems given at the end of Section 2, *e.g.*, the secrecy and the weak plan compliance problems. We circumvent this problem by showing how to reuse obsolete constants instead of creating new values.

Consider as an intuitive example the scenario where customers are waiting at a counter. Whenever a new customer arrives, he picks a number and waits until his number is called. Since only one person is called at a time, usually in a first come first serve fashion, a number that is picked has to be a fresh value, that is, it should not belong to any other customer in the waiting room. Furthermore, since only a bounded number of customers wait at the counter in a period of time, one only needs a bounded number of tickets: once a customer is finished, his number can be in fact reused and assigned to another customer.

We generalize the idea illustrated by the example above to systems with balanced actions. Since in such systems all configurations have the same number of facts and the size of facts is bounded, in practice we do not need an unbounded number of new constants in order to reach a goal, but just a small number of them.

Whenever a nonce needs to be created, we will pick a nonce name from the fixed set of nonce names in such a way that it differs from all the nonce names in the given configuration. Although nonce names have been fixed in advance, they can be considered fresh. We call actions that pick fresh values from a small set of nonces *guarded nonce generation*.

Consequently, in a given planning problem we only need to consider a small number of nonces names. This is formalized by the following theorem.

Theorem 4.1. *Given an LSTS with balanced actions that can create nonces, any plan leading from an initial configuration W to a partial goal Z can be transformed into another plan also leading from W to Z that uses only a polynomial number of nonces, $2mk$, with respect to the number of facts, m, in W and an upper bound on the size of facts, k.*

The proof of Theorem 4.1 relies on the observation that from the perspective of an insider of the system two configurations can be considered the same whenever they only differ on the names of the nonces used.

Consider for example the following two configurations, where the n_is are nonces and t_is are constants in the initial alphabet:

$\{F_A(t_1, n_1), G_B(n_2, n_1), H_{pub}(n_3, t_2)\}$ and $\{F_A(t_1, n_4), G_B(n_5, n_4), H_{pub}(n_6, t_2)\}$

Since these configurations only differ in the nonce's names used, they can be regarded as equivalent: the same fresh value, n_1 in the former configuration and n_4 in the latter, is shared by the agents A and B, and similarly, for the new values n_2 and n_5, and n_3 and n_6. Inspired by a similar notion in λ-calculus [13], we say that these configurations above are α-equivalent.

Definition 4.2. Two *configurations* $\mathcal{S}_1$ and $\mathcal{S}_2$ are *α-equivalent*, denoted by $\mathcal{S}_1 =_\alpha \mathcal{S}_2$, if there is a bijection σ that maps the set of all nonce names appearing in one configuration to the set of all nonce names appearing in the other configuration, such that the set $\mathcal{S}_1\sigma = \mathcal{S}_2$.

The two configurations given above are α-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 relation $=_\alpha$ is indeed an equivalence, that is, it is symmetric, transitive, and reflexive.

The following lemma formalizes the intuition described above that from the point of view of an insider two α-equivalent configurations are the same, that is, one can apply the same action to one or the other and the resulting configurations are also equivalent. This is similar to the notion of bisimulation in process calculi [37].

Lemma 4.3. *Let m be the number of facts in a configuration $\mathcal{S}_1$ and k be an upper bound on the size of facts. Let $\mathcal{N}_{m,k}$ be a fixed set of $2mk$ nonce names. Suppose that the configuration S_1 is α-equivalent to a configuration S_1' and, in addition, each of the nonce names occurring in S_1' belongs to $\mathcal{N}_{m,k}$. Let an instance of the action r transform the configuration S_1 into the configuration S_2. Then there is a configuration S_2' such that: (1) an instance of action r transforms S_1' into S_2'; (2) S_2' is α-equivalent to S_2; and (3) each of the nonce names occurring in S_2' belongs to $\mathcal{N}_{m,k}$.*

Proof We alter the given transformation $S_1 \rightarrow_r S_2$, which can in principle include nonce creation, into $S_1' \rightarrow_{r'} S_2'$ so that the action r' is an action of guarded nonce generation. It does not create new values, instead it chooses nonce names from the set $\mathcal{N}_{m,k}$, in such a way that the chosen nonce names differ from any values in the enabling configuration S_1'. Although these names have been fixed in advance, they can be considered fresh.

Let r be a balanced action that does not create nonces. Assume an instance of r is used to transform $\mathcal{S}_1$ to $\mathcal{S}_2$ and assume that the nonces in $\mathcal{S}_1$ are $\vec{n}$. Let σ be a bijection between the nonces of $\mathcal{S}_1$ and $\mathcal{S}_1'$. Then an instance of r where the nonces $\vec{n}$ are replaced with $\vec{n}\sigma$ transforms the configuration $\mathcal{S}_1'$ into $\mathcal{S}_2'$. Configurations $\mathcal{S}_2'$ and $\mathcal{S}_2$ are α-equivalent since these configurations differ only in nonce names, as per bijection σ.

It is more interesting when a rule r creates nonces $\vec{n_2}$ resulting in S_2. Since the number of all places (slots for values) in a configuration is bounded by mk, we can find enough elements $\vec{n_2}'$ (at most mk in the extreme case where all nonces are supposed to be created simultaneously) in the set of $2mk$ nonce names, $\mathcal{N}_{m,k}$, that do not occur in $\mathcal{S}_1'$. Values $\vec{n_2}'$ can therefore be considered fresh and used instead of $\vec{n_2}$. Let δ be the bijection between nonce names $\vec{n_2}$ and $\vec{n_2}'$ and let σ be a bijection between the nonces of $\mathcal{S}_1$ and $\mathcal{S}_1'$. Then the action $r' = r\delta\sigma$ of guarded nonce creation is an instance of action r which is enabled in configuration S_1' resulting in configuration S_2'. Configurations S_2 and S_2' are α-equivalent because of the bijection $\delta\sigma$.

Moreover, from the assumption that goal and critical configurations are closed under renaming of nonces, it follows that if the configuration S_2 is a goal configuration, then S_2' is also a goal configuration. Similarly, if S_2 is not a critical configuration, then the configuration S_2' is also not critical. □

We are now ready to prove Theorem 4.1:

Proof (of Theorem 4.1). The proof is by induction on the length of a plan and it is based on Lemma 4.3. Let T be an LSTS with balanced actions that can create

nonces, m the number of facts in the initial configuration, and k the bound on size of each fact. Let $\mathcal{N}_{m,k}$ be a fixed set of $2mk$ nonce names. Given a plan P leading from the initial configuration W to a partial goal Z, we adjust it so that all nonces along the plan P' are taken from $\mathcal{N}_{m,k}$. Notice that since all actions are balanced, the size of all configurations in P are the same as the size of W, namely m.

For the base case, assume that the plan is of the length 0, that is, the configuration W already contains Z. Since we assume that goal and initial configurations are closed under renaming of nonces, we can rename the nonces in W by nonces from $\mathcal{N}_{m,k}$.

Assume that any plan of length n can be transformed into a plan that uses the fixed set of nonce names. Let a plan P of the length $n+1$ be such that $W \rhd_T^* ZU$. Let r be the last action in P and $Z_1 \rhd_r ZU$. By induction hypothesis we can transform the plan $W \rhd_T^* Z_1$ into a plan $W' \rhd_T^* Z_1'$, with all configurations α-equivalent to corresponding configurations in the original plan, such that it only contains nonces from the set $\mathcal{N}_{m,k}$.

We can then apply Lemma 4.3 to configurations Z_1, Z_1' and the action r to conclude that there is a configuration $Z'U'$ that is α-equivalent to configuration ZU such that all the nonces in the configuration $Z'U'$ belong to $\mathcal{N}_{m,k}$. Therefore, all the nonces contained in the transformed plan P', *i.e.* in the plan $W' \rhd_T^* Z'U'$ are taken from $\mathcal{N}_{m,k}$.

Notice that since ZU is the goal configuration, so is the configuration $Z'U'$, because of the assumption that goal configurations are closed under nonce renaming and $ZU =_\alpha Z'U'$. Also notice that no critical configuration is reached in the new plan because corresponding configurations from these plans are α-equivalent, and we assume that critical configurations are closed under renaming of nonce names. □

Corollary 4.4. *For LSTSes with balanced actions that can create nonces, we only need to consider the reachability problem with a polynomial number of* fresh *values, which can be fixed in advance, with respect to the number of facts in the initial configuration and the upper bound on the size of facts.*

Notice that, since plans can be of exponential length, a nonce name from $\mathcal{N}_{m,k}$ can, in principal, be used in guarded nonce creation an exponential number of times. However, each time a nonce name is used, it appears fresh with respect to the enabling configuration.

Table 1: Summary of the complexity results for the secrecy, weak plan, system, and plan compliance problems. We mark the new results appearing here with a $\star$. We also show here that the complexity for the system compliance problem when actions are possibly unbalanced and can create fresh values is undecidable.

Compliance Problems	Balanced Actions		Possibly unbalanced actions and no nonces
	No fresh values	Possible nonces	
Secrecy	PSPACE-complete [32]	PSPACE-complete*	Undecidable [19]
Weak Plan	PSPACE-complete [32]	PSPACE-complete*	Undecidable [31]
System	PSPACE-complete [32]	PSPACE-complete*	EXPSPACE-complete [31]; Undecidable with nonces [19]
Plan	PSPACE-complete [32, 45]	PSPACE-complete*	Undecidable [31]

5. Complexity Results

In this Section we discuss complexity results for the planning problems introduced in Section 2, namely, the weak plan compliance problem, the plan compliance problem, the system compliance problem and the secrecy problem. Table 1 summarizes the complexity results for these compliance problems.

We start, mainly for completeness, with the simplest form of systems, namely, those that contain only actions of the form $a \to a'$, called *context-free monadic actions*, which only change a single fact from a configuration. The following result can be inferred from [19, Proposition 5.4].

Theorem 5.1. *Given an LSTS with only actions of the form $a \to a'$, the weak plan compliance, the plan compliance problem, and the secrecy problems are in P.*

Our next result improves the result in [32, Theorem 6.1] since any type of balanced actions was allowed in that encoding. Here, on the other hand, we allow only *monadic actions*, which are actions of the form $ab \to a'b$, *i.e.*, balanced actions that can modify at most a single fact and in the process check whether a fact is present in the configuration. We tighten the lower bound by showing that all the decision problems described in Section 2 for LSTSes with monadic actions are also PSPACE-hard. The main challenge here is to simulate operations over a

non-commutative structure by using a commutative one, namely, to simulate the behavior of a Turing machine that uses a sequential, non-commutative tape in our formalism that uses commutative multisets.

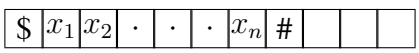
Theorem 5.2. *Given an LSTS $\mathcal{T}$ with only actions of the form $ab \rightarrow a'b$, then the problems of weak plan compliance, plan compliance, system compliance and the secrecy problem are PSPACE-hard in the size of $\mathcal{T}$.*

The PSPACE upper bound for this problem can be inferred directly from [32].

Proof We start the proof with the weak plan compliance problem. In order to prove the lower bound, we encode a non-deterministic Turing machine $\mathcal{M}$ that accepts in space n within actions of the form $ab \rightarrow a'b$, whenever each of these actions is allowed any number of times. In our proof, we do not use critical configurations and need just one agent A. Without loss of generality, we assume the following:

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

(b) Initially, an *input* string, say $x_1x_2 \ldots x_n$, is written in cells 1, 2,..., n on the tape. In addition, a special marker # is written in the $(n+1)$-th cell.



(c) The program of $\mathcal{M}$ contains no instruction that could erase either \$ or #. There is no instruction that could move the head of $\mathcal{M}$ either to the right when $\mathcal{M}$ scans symbol #, or to the left when $\mathcal{M}$ scans symbol \$. As a result, $\mathcal{M}$ acts in the space between the two unerased markers.

(d) Finally, $\mathcal{M}$ has only one *accepting* state q_f, and, moreover, all *accepting* configurations in space n are of one and the same form.

For each n, we design a local state transition system T_n as follows:
First, we introduce the following propositions: $R_{i,\xi}$ which denotes that *"the i-th cell contains symbol ξ"*, where $i = 0, 1, \ldots, n+1$, ξ is a symbol of the tape alphabet of $\mathcal{M}$, and $S_{j,q}$ which denotes that *"the j-th cell is scanned by $\mathcal{M}$ in state q"*, where $j = 0, 1, \ldots, n+1$, q is a state of $\mathcal{M}$.
Given a *machine configuration* of $\mathcal{M}$ in space n - that $\mathcal{M}$ scans j-th cell in state q, when a string $\xi_0\xi_1\xi_2 \ldots \xi_i \ldots \xi_n\xi_{n+1}$ is written left-justified on the otherwise blank

tape, we will represent it by a configuration of T_n of the form (here ξ_0 and ξ_{n+1} are the end markers):

$$S_{j,q}R_{0,\xi_0}R_{1,\xi_1}R_{2,\xi_2}\cdots R_{n,\xi_n}R_{n+1,\xi_{n+1}}. \tag{2}$$

Second, each instruction γ in $\mathcal{M}$ of the form $q\xi \to q'\eta D$, denoting *"if in state q looking at symbol ξ, replace it by η, move the tape head one cell in direction D along the tape, and go into state q'"*, is specified by the set of $5(n+2)$ actions of the form:

$$\begin{array}{lll} S_{i,q}R_{i,\xi} \to_A F_{i,\gamma}R_{i,\xi}, & F_{i,\gamma}R_{i,\xi} \to_A F_{i,\gamma}H_{i,\gamma}, & F_{i,\gamma}H_{i,\gamma} \to_A G_{i,\gamma}H_{i,\gamma}, \\ G_{i,\gamma}H_{i,\gamma} \to_A G_{i,\gamma}R_{i,\eta}, & G_{i,\gamma}R_{i,\eta} \to_A S_{i_D,q'}R_{i,\eta}, & \end{array} \tag{3}$$

where $i=0,1,\ldots,n+1$, $F_{i,\gamma}$, $G_{i,\gamma}$, $H_{i,\gamma}$ are auxiliary atomic propositions, $i_D := i+1$ if D is *right*, $i_D := i-1$ if D is *left*, and $i_D := i$, otherwise.

The idea behind this encoding is that by means of such five monadic rules, applied in succession, we can simulate any successful non-deterministic computation in space n that leads from the initial configuration, W_n, with a given input string $x_1x_2\ldots x_n$, to the accepting configuration, Z_n.

The *faithfulness* of our encoding heavily relies on the fact that any machine configuration includes exactly one machine state q. Because of the specific form of our actions in (3), any configuration reached along a plan $\mathcal{P}$, leading from W_n to Z_n, has exactly one occurrence of either $S_{i,q}$ or $F_{i,\gamma}$ or $G_{i,\gamma}$. Therefore the actions in (3) are necessarily used one after another as below:

$$S_{i,q}R_{i,\xi} \to_A F_{i,\gamma}R_{i,\xi} \to_A F_{i,\gamma}H_{i,\gamma} \to_A G_{i,\gamma}H_{i,\gamma} \to_A G_{i,\gamma}R_{i,\eta} \to_A S_{i_D,q'}R_{i,\eta}.$$

Moreover, any configuration reached by using the plan $\mathcal{P}$ is of the form similar to (2), and, hence, represents a configuration of $\mathcal{M}$ in space n.
Passing through this plan $\mathcal{P}$ from its last action to its first v_0, we prove that whatever intermediate action v we take, there is a successful non-deterministic computation performed by $\mathcal{M}$ leading from the configuration reached to the *accepting* configuration represented by Z_n. In particular, since the first configuration reached by $\mathcal{P}$ is W_n, we can conclude that the given input string $x_1x_2\ldots x_n$ is accepted by $\mathcal{M}$.

By the above encoding we reduce the problem of a Turing machine acceptance in n- space to a weak plan compliance problem with no critical configurations and conclude that the weak plan compliance problem is PSPACE-hard.

The secrecy problem is a special case of the weak plan compliance problem with no critical configurations and with the goal configuration having a negative

connotation of intruder learning the secret. To the above encoding we add the action $S_{i,q_f} \to M_s(s)$, for the accepting state q_f and the constant s denoting the secret. This action reveals the secret to the intruder. Consequently, the secrecy problem is also PSPACE-hard.

Finally, since the encoding involves no critical configurations both the plan compliance and the system compliance problem are also PSPACE-hard. □

In order to obtain a faithful encoding, one must be careful, especially, with commutativity. If we attempt to encode these actions by using, for example, the following four monadic actions

$$\begin{array}{ll} S_{i,q}R_{i,\xi} \to_A F_{i,\gamma}R_{i,\xi}, & F_{i,\gamma}R_{i,\xi} \to_A F_{i,\gamma}H_{i,\gamma}, \\ F_{i,\gamma}H_{i,\gamma} \to_A F_{i,\gamma}R_{i,\eta}, & F_{i,\gamma}R_{i,\eta} \to_A S_{i_D,q'}R_{i,\eta}, \end{array}$$

then such encoding would not be faithful because of the following conflict:

$$(F_{i,\gamma}R_{i,\xi} \to_A F_{i,\gamma}H_{i,\gamma}) \quad \text{and} \quad (F_{i,\gamma}R_{i,\eta} \to_A S_{i_D,q'}R_{i,\eta}).$$

Also notice that one cannot always use a set of five monadic actions similar to those in (3) to faithfully simulate non-monadic actions of the form $ab \to cd$. Specifically, one cannot always guarantee that a goal is reached after all five monadic actions are used, and not before. For example, if our goal is to reach a configuration containing the fact c and we consider a configuration containing both c and d as critical, then with the monadic rules it would be possible to reach a goal without reaching a critical state, whereas, when using the non-monadic action, one would not be able to do so. This is because, when applying the action $ab \to cd$, one necessarily reaches a critical state. However, in the encoding of Turing machines used in the proof above, this is not a problem since all propositions of the form $S_{i,q}$ do not appear in the intermediate steps, as illustrated above.

LSTSes that can create nonces. We turn our attention to the case when actions can create nonces. We show that the problems of the weak plan compliance, plan compliance and system compliance as well as the secrecy problem for LSTSes with balanced actions that can create nonces are in PSPACE. Combining this upper bound with the lower bound given in Theorem 5.2, we can infer that this problem is indeed PSPACE-complete.

Recall that, in Section 4 we introduce a formalization of freshness in balanced systems. Instead of (proper) nonce creation, in balanced systems we consider *guarded nonce creation*, see Lemma 4.3. We are therefore able to simulate plans that include actions of nonce creation with plans containing α-equivalent configurations such that the whole plan includes only a small number of nonce names,

polynomial in the size of the configurations and in the bound on size of facts. This is an important assumption in all of our results related to balanced systems.

To determine the existence of a plan we only need to consider plans that never reach α-equivalent configurations more than once. If a plan loops back to a configuration α-equivalent to a previously reached configuration, there is a cycle of actions which could have been avoided. The following lemma provides an upper bound on the number of different configurations given an initial finite alphabet.

Lemma 5.3. *Given an LSTS T under a finite alphabet Σ, then the number of configurations, $L_T(m, k)$, that are pairwise not α-equivalent and whose number of facts (counting repetitions) is exactly m is such that*

$$L_T(m, k) \leq J^m(D + 2mk)^{mk},$$

where J and D are, respectively, the number of predicate symbols and the number of constant and function symbols in the initial alphabet Σ, and k is an upper bound on the size of facts.

Proof Since a configuration contains m facts and each fact can contain only one predicate symbol, there are m slots for predicate names. Moreover, since the size of facts is bounded by k, there are at most mk slots in a configuration for constants and function symbols. Constants can be either constants in the initial alphabet Σ or nonce names. However, following Theorem 4.1, we need to consider only $2mk$ nonces. Hence, there at most $J^m(D + 2mk)^{mk}$ configurations that are not α-equivalent, where J and D are, respectively, the number of predicate symbols and the number of constant and function symbols in the initial alphabet Σ. □

Clearly, the above upper bound on the number of configurations is an overestimate. It does not take into account, for example, the equivalence of configurations that only differ on the order of the facts. For our purposes, however, it will be enough to assume such a bound.

Although the secrecy problem as well as the weak plan compliance, plan compliance and system compliance problems are stated as decision problems, we prove more than just PSPACE decidability. Ideally we would also be able to generate a plan in PSPACE when there is a solution. Unfortunately, as we have illustrated in Section 3, the number of actions in the plan may already be exponential in the size of the inputs precluding PSPACE membership of plan generation. These plans could, in principle, also involve an exponential number of nonces, as discussed at the end of Section 4. For the reason above we follow [32] and use the notion of "scheduling" a plan, in which an algorithm will also take an input i and output the i-th step of the plan.

Definition 5.4. An algorithm is said to *schedule* a plan for the given planning problem if it (1) finds a plan if one exists, and (2) on input i, if the plan contains at least i actions, then it outputs the i^{th} action of the plan, otherwise it outputs *no*.

Following [32], we assume that for a given LSTS, there are three programs, $\mathcal{C}, \mathcal{G}$, and $\mathcal{T}$, such that they return the value 1 in polynomial space when given as input, respectively, a configuration that is critical, a configuration that contains the goal configuration, and a transition that is valid, that is, an instance of an action in the LSTS, and return 0 otherwise. For the secrecy problem, we need to additionally assume a program $\mathcal{M}$ that returns the value 1 in polynomial space when given as input a rule from the intruder's theory, and return 0 otherwise. Later in Section 6 we give an example of an intruder theory.

The following theorem establishes the PSPACE upper bound for the weak plan compliance and secrecy problems.

Theorem 5.5. *Given an LSTS T with balanced actions that can create nonces and an intruder theory M, then the weak plan compliance problem and the secrecy problem are in PSPACE in the following parameters:*

- *the size, m, of the initial configuration W,*
- *bound on the size of facts, k,*
- *the size of the programs $\mathcal{G}, \mathcal{C}$, $\mathcal{T}$, and $\mathcal{M}$, described above, and*
- *a natural number $0 \leq i \leq L_T(m, k)$.*

Proof For both decision problems, we rely on the fact that NPSPACE, PSPACE, and co-PSPACE are the same complexity class [47]. We first prove that the weak plan compliance problem is in PSPACE. We modify the algorithm proposed in [32] in order to accommodate the creation of nonces. The algorithm returns "yes" whenever there is compliant plan from the initial configuration W to a goal configuration. Our algorithm non-deterministically searches whether a goal configuration *is reachable*, that is, a configuration S such that $\mathcal{G}(S) = 1$, without passing through a critical configuration. Then we apply Savitch's Theorem [47] to determinize this algorithm.

The algorithm begins with $W_0 := W$. For any $t \geq 0$, we first check if $\mathcal{C}(W_t) = 1$. If this is the case, then the algorithm outputs "no." We also check whether the configuration W_t is a goal configuration, that is, if $\mathcal{G}(W_t) = 1$. If so, we end the algorithm by returning "yes." Otherwise, we guess an action r such

that $\mathcal{T}(r) = 1$ and that is applicable using the configuration W_t. If no such action exists, then the algorithm outputs "no." Otherwise, we replace W_t by the configuration W_{t+1} resulting from applying the action r to W_t. Following Lemma 5.3 we know that a goal configuration is reached if and only if it is reached in $L_T(m,k)$ steps. We use a global counter, called step-counter, to keep track of the number of actions used in a potential plan constructed by this algorithm.

As pointed out in Section 3, plans can, in principle, use an exponential number of fresh values. However, as we have shown before in Section 4, it is enough to use a set with only $2mk$ nonce names. This set of nonce names is not related to any particular plan, but is fixed in advance. Then whenever an action creates a fresh value, we can search for names in this set that are different from any constants in the enabling configuration, that is, a fresh value. This process is shown in the proof of Theorem 4.1.

We now show that this algorithm runs in polynomial space. We start with the step-counter: The greatest number reached by this counter is $L_T(m,k)$. When stored in binary encoding, this number takes only space polynomial to the given inputs:

$$\begin{aligned}\log_2(L_T(m,k)) &\leq \log_2(J^m(D+2mk)^{mk}) = \log_2(J^m) + \log_2((D+2mk)^{mk}) \\ &= m\log_2(J) + mk\log_2(D+2mk).\end{aligned}$$

Therefore, one only needs polynomial space to store the values in the step-counter. Following Theorem 4.1 there are at most polynomially many nonces used in a run, namely at most $2mk$. Hence nonces can also be stored in polynomial space.

We must also be careful to check that any configuration, W_t, can be stored in polynomial space with respect to the given inputs. Since our system is balanced and we assume that the size of facts is bounded, the size of a configuration remains the same throughout the run. Finally, the algorithm needs to keep track of the action r guessed when moving from one configuration to another and for the scheduling of a plan. It has to store the action that has been used at the i^{th} step. Since any action can be stored by remembering two configurations, one can also store these actions in space polynomial to the inputs.

A similar algorithm can be used for the secrecy problem. The only modification to the previous algorithm is that one does not need to check for critical configurations as in the secrecy problem there are no such configurations. □

Theorem 5.6. *Given an LSTS with balanced actions that can create nonces, then the system compliance problem is in PSPACE in the following parameters:*

- *the size, m, of the initial configuration W,*
- *bound on the size of facts, k,*
- *the size of the programs $\mathcal{G}, \mathcal{C}$, and $\mathcal{T}$ and*
- *a natural number $0 \leq i \leq L_T(m, k)$.*

Proof In order to show that the system compliance problem is in PSPACE we modify the algorithm proposed in [32] to accommodate the nonce creation. Again we rely on the fact that NPSPACE, PSPACE, and co-PSPACE are the same complexity class [47]. We use the same notation from the proof of Theorem 5.5 and make the same assumptions.

Following Theorem 4.1 we can accommodate nonce creation by replacing the relevant nonce occurrence(s) with nonces from a fixed set, so that they are different from any of the nonces in the enabling configuration. As before, this set of $2mk$ nonce names is not related to a particular plan, but fixed in advance for a given LSTS, where m is the number of facts in the configuration of the system and k is the bound on the size of the facts.

We first need to check that none of the critical configurations are reachable from W. To do this we provide a non-deterministic algorithm which returns "yes" exactly when a critical configuration is reachable. The algorithm starts with $W_0 := W$. For any $t \geq 0$, we first check if $\mathcal{C}(W_t) = 1$. If this is the case, then the algorithm outputs "yes". Otherwise, we guess an action r such that $\mathcal{T}(r) = 1$ and that it is applicable in the configuration W_t. If no such action exists, then the algorithm outputs "no". Otherwise, we replace W_t by the configuration W_{t+1} resulting from applying the action r to W_t. Following Lemma 5.3 we know that at most $L_T(m, k)$ guesses are required, and therefore we use a global step-counter to keep track of the number of actions. As shown in the proof of Theorem 5.5, the value of this counter can be stored in PSPACE.

Next we apply Savitch's Theorem to determinize the algorithm. Then we swap the accept and fail conditions to get a deterministic algorithm that accepts exactly when all critical configurations are unreachable.

Finally, we have to check for the existence of a compliant plan. For that we apply the same algorithm as for the weak plan compliance problem from Theorem 5.5, skipping the checking of critical states since we have already checked that none of the critical configurations are reachable from W. From what has been shown above we conclude that the algorithm runs in polynomial space. Therefore the system compliance problem is in PSPACE. □

Next we turn to the plan compliance problem for systems with balanced actions that can create nonces. In addition to avoiding critical configurations, a compliant plan also guarantees to every agent that, as long as he follows the plan, the other agents cannot collude to reach a configuration critical for him. Agents are therefore assured that in case they drop from the collaboration for any reason, others cannot violate their confidentiality policies. As soon as one agent deviates from the plan, the other agents may choose to stop their participation. They can do so with the assurance that the remaining agents will never be able to reach a configuration critical for those agents that quit the collaboration.

The plan compliance problem can be re-stated as a weak plan compliance problem with a larger set of configurations that should be avoided, called *semi-critical configurations*. Intuitively, a semi-critical configuration for an agent A is a configuration from which a critical configuration for A could be reached by the other participants of the system without the participation of A. Therefore in the plan compliance problem, a compliant plan not only avoids critical configurations, but also avoids configurations that are semi-critical. Hence, the plan compliance problem is the same as the weak plan compliance problem when considering critical both the original critical configurations of the problem as well as the semi-critical configurations of any agent.

Definition 5.7. A configuration X is *semi-critical for an agent* A in the given planning problem if a configuration Y that is critical for A is reachable from X using the actions belonging to all agents except to A, *i.e.*, if $X \rhd^*_{-A} Y$. A configuration is simply called *semi-critical* if it is semi-critical for some agent of the system.

We will follow this intuition and construct an algorithm for the plan compliance problem similar to the one used for the weak plan compliance problem, that will include a sub-procedure that checks if a configuration is semi-critical for an agent.

Theorem 5.8. *Given an LSTS with balanced actions that can create nonces, then the plan compliance problem is in PSPACE in the following parameters:*

- *the size, m, of the initial configuration W,*
- *bound on the size of facts, k,*
- *the size of the programs $\mathcal{G}, \mathcal{C}$, and $\mathcal{T}$ and*

- *a natural number* $0 \leq i \leq L_T(m,k)$.

Proof The proof is similar to the proof of Theorem 5.5 and the proof of the PSPACE result of the plan compliance for balanced systems in [45]. Again we rely on the fact that NPSPACE, PSPACE, and co-PSPACE are one and the same complexity class.

Assume as inputs an initial configuration W containing m facts, an upper bound on the size of facts k, a natural number $0 \leq i \leq L_T(m,k)$, and programs $\mathcal{G}, \mathcal{C}$, and $\mathcal{T}$ that run in polynomial space and that are slightly different to those in Theorem 5.5. This is because for plan compliance it is important to know as well to whom an action belongs to and similarly for which agent a configuration is critical. Program $\mathcal{T}$ recognizes actions of the system so that $\mathcal{T}(j,r) = 1$ when r is an instance of an action belonging to agent A_j and $\mathcal{T}(j,r) = 0$ otherwise. Similarly, program $\mathcal{C}$ recognizes critical configurations so that $\mathcal{C}(j,Z) = 1$ when configuration Z is critical for agent A_j and $\mathcal{C}(j,Z) = 0$ otherwise. Program $\mathcal{G}$ is the same as described earlier, *i.e.*, $\mathcal{G}(Z) = 1$ if Z contains a goal and $\mathcal{G}(Z) = 0$ otherwise.

First we construct the algorithm ϕ that checks if a configuration is semi-critical for an agent. While guessing the actions of a compliant plan at each configuration Z reached along the plan we need to check whether for any agent A_j other agents could reach a configuration critical for A_j. More precisely, at configuration Z, for an agent A_j and $Z_0 = Z$, the following nondeterministic algorithm looks for configurations that are semi-critical for the agent A_j:

1. Check if $\mathcal{C}(j, Z_t) = 1$, then ACCEPT; otherwise continue;
2. Guess an action r and an agent $A_l \neq A_j$ such that $\mathcal{T}(l,r) = 1$ and that r is enabled in configuration Z_t; if no such action exists then FAIL;
3. Apply r to Z_t to get configuration Z_{t+1}.

After guessing $L_T(m,k)$ actions, if the algorithm has not yet returned anything, it returns FAIL. We can then reverse the accept and reject conditions and use Savitch's Theorem to get a deterministic algorithm $\phi(j,Z)$ which accepts if every configuration V satisfying $Z \rhd^*_{-A_j} V$ also satisfies $\mathcal{C}(j,V) = 0$, and rejects otherwise. In other words, $\phi(j,Z)$ accepts only in the case when Z is not semi-critical for agent A_j. Next we construct the deterministic algorithm $\mathcal{C}'(Z)$ that accepts only in the case when Z is not semi-critical simply by checking if $\phi(j,Z)$ accepts for every j; if that is the case $C'(Z) = 1$, otherwise $C'(Z) = 0$.

Now we basically approach the weak plan compliance problem considering all semi-critical configurations as critical by using the algorithm from the proof of Theorem 5.5 with the $\mathcal{C}'$ as the program that recognizes the critical configurations.

We now show that algorithm $\mathcal{C}'$ runs in polynomial space.

Following Theorem 4.1 we can accommodate nonce creation in polynomial space by replacing the relevant nonce occurrence(s) with nonces from a fixed set of $2mk$ nonce names, so that they are different from any of the nonces in the enabling configuration.

The algorithm ϕ stores at most two configurations at a time which are of the constant size, same size the initial configuration W. Also, the action r can be stored with two configurations. At most two agent names are stored at a time. Since the number of agents n is much smaller than the size of the configuration m, simply by the nature of our system, we can store each agent in space $\log n$. As in the proof of Theorem 5.5 only a polynomial space is needed to store the values in the step-counter, even though the greatest number reached by the step counter is $L_T(m, k)$, which is exponential in the given inputs. Since checking whether $\mathcal{C}(j, Z_t) = 1$ and $\mathcal{T}(l, r) = 1$ can be done in space polynomial to $|W|$, $|\mathcal{C}|$ and $|\mathcal{T}|$, algorithm ϕ, and consequently $\mathcal{C}'$, work in space polynomial to the given inputs.

We combine this with Theorem 5.5 to conclude that the plan compliance problem is in PSPACE. □

Given the PSPACE lower bound for the secrecy, weak plan compliance, system compliance, and the plan compliance problem in Theorem 5.2 and the PSPACE upper bound given in the theorems above, we can conclude that all these problems are PSPACE-complete.

Discussion on related work. This PSPACE-complete result contrasts with results in [19], where the secrecy problem is shown to be undecidable. Although in [19] an upper bound on the size of facts was imposed, the actions were not restricted to be balanced. Therefore, in [19] it was possible for the intruder to remember an unbounded number of facts, while here the memory of all agents is bounded. Moreover, for the DEXP result in [19], a constant bound on the number of nonces that can be created was imposed, whereas such a bound is not imposed here.

We also point out that our PSPACE upper bounds improve the PSPACE upper bounds in [32, 30] by not only allowing actions that can create fresh values, but also in that we consider the size of facts as an input bound, whereas [32, 30] consider the size of facts a fixed bound.

Complexity of possibly unbalanced LSTSes. For LSTSes with possibly unbalanced actions that cannot create fresh values, it was shown in [31] that the complexity of both the weak plan and the plan compliance problems is undecidable,

while the complexity of the system compliance problem is EXPSPACE-complete. Given these results we can immediately infer that the complexity of the weak plan and plan compliance are also undecidable when we allow actions to create fresh values. Next, we show that when actions are possibly unbalanced and can create fresh values, then the system compliance problem is also undecidable.

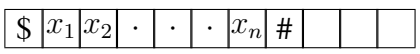
Theorem 5.9. *The system compliance problem for general LSTSes with actions that can create fresh values is undecidable.*

Proof The proof relies on undecidability of acceptance of Turing machines with unbounded tape. The proof is similar to the undecidability proof of multiset rewrite rules with existential quantifiers in [19].

Without loss of generality, we assume the following:

(a) $\mathcal{M}$ has only one tape, which is one-way infinite to the right. The leftmost cell contains the marker \$.

(b) Initially, an input string, say $x_1 x_2 \ldots x_n$, is written in cells 1, 2,. . . , n on the tape. In addition, a special marker # is written in the $(n+1)$-th cell.



(c) The program of $\mathcal{M}$ contains no instruction that could erase \$. There is no instruction that could move the head of $\mathcal{M}$ to the left when $\mathcal{M}$ scans symbol \$ and in case when $\mathcal{M}$ scans symbol #, tape is adjusted, *i.e.* another cell is inserted so that $\mathcal{M}$ scans symbol a_0 and the cell immediately to the right contains the symbol #.

(d) Finally, $\mathcal{M}$ has only one accepting state q_f.

Given a Turing machine $\mathcal{M}$ we construct an LSTS $T_{\mathcal{M}}$ with actions that can create fresh values. The alphabet of $T_{\mathcal{M}}$ has four sorts: $state$ for the Turing machine states, $cell$ and $nonce < cell$ for the cell names, and $symbol$ for the cell contents. We introduce constants $a_0, a_1, \ldots, a_m : symbol$ to represent symbols of the tape alphabet with a_0 denoting blank; constants $q_0, q_1, \ldots, q_f : state$ for the machine states, where q_0 is the initial state and q_f is the accepting state; and finally constants $\$, c_1, \ldots, c_n, \# : cell$ for the names of the cells including the leftmost cell \$ denoting the beginning of the tape and the rightmost cell # denoting end of tape.

Predicates $Curr : state \times cell$, $Cont : cell \times symbol$ and $Adj : cell \times cell$ denote, respectively, the current state and tape position, the contents of the cells, and the adjacency between the cells.
The tape maintenance is formalized by the following action:

$$Adj(c, \#) \rightarrow \exists c'.Adj(c, c')\ Adj(c', \#)\ Cont(c', \#)\,. \tag{4}$$

By using this actions one is able to extend the tape by labeling the new cell with a fresh value, c'. Notice that due to the rule above, one needs an unbounded number of fresh values since an unbounded number of cells can be used. To each machine instruction $q_i a_s \rightarrow q_j a_t L$ denoting "if in state q_i looking at symbol a_s, replace it by a_t, move the tape head one cell to the left and go into state q_j" we associate action:

$$Curr(q_i, c)\ Cont(c, a_s)\ Adj(c', c) \rightarrow Curr(q_j, c')\ Cont(c, a_t)\ Adj(c', c). \tag{5}$$

Notice that we move to the left by using the fact $Adj(c', c)$ denoting that the cell c' is to the cell immediately to the left of the cell c. Similarly, to each machine instruction $q_i a_j \rightarrow q_s a_t R$ denoting "if in state q_i looking at symbol a_s, replace it by a_t, move the tape head one cell to the right and go into state q_j" we associate action:

$$Curr(q_i, c)\ Cont(c, a_s)\ Adj(c, c') \rightarrow Curr(q_j, c')\ Cont(c, a_t)\ Adj(c, c')\,. \tag{6}$$

This action assumes that there is an available tape cell to the right of the tape head. If this is not the case, one has to first use the action (4) that creates a new cell in the tape and only then apply the action (6).

Given a machine configuration of $\mathcal{M}$, where $\mathcal{M}$ scans cell c in state q, when a string $\$x_1x_2\ldots x_k\#$ is written left-justified on the otherwise blank tape, we represent it by the following initial configuration of $T_{\mathcal{M}}$

$$\begin{array}{c} Cont(c_0, \$)\ Cont(c_1, x_1)\ldots\ Cont(c_k, x_k)\ Cont(c_{k+1}, \#) \\ Curr(q, c)\ Adj(c_0, c_1)\ \ldots, Adj(c_k, c_{k+1})\,. \end{array} \tag{7}$$

The goal configuration is the one containing the fact $Curr(q_f, c)$.

The *faithfulness* of our encoding relies on the fact that any machine configuration includes exactly one machine state q. This is because of the specific form of actions (4), (5) and (6), which enforce that any reachable configuration has exactly one occurrence of $Curr(q, c)$. Moreover, any reachable configuration is of the form similar to (7), and, hence, represents a configuration of $\mathcal{M}$.

Passing through the plan $\mathcal{P}$ from the initial configuration W to the goal configuration Z, from its last action to its first r_0, we prove that whatever intermediate action r we take, there is a successful non-deterministic computation performed by $\mathcal{M}$ leading from the configuration reached to the *accepting* configuration represented by Z. In particular, since the first configuration reached by $\mathcal{P}$ is W, we can conclude that the given input string $x_1x_2 \ldots x_n$ is accepted by $\mathcal{M}$.

Notice that the above encoding involves no critical configurations so we achieve undecidability already for that simplified case. Consequently we get undecidability of LSTSes with actions that can create nonces for all three types of compliances. □

6. Application: Protocol theories with bounded memory intruder

This section enters into the details of whether malicious agents, or intruders, with the same capabilities as the other agents are able to discover some secret information. In particular, we modify the intruder theory in [19] to our setting where all agents, including the intruder, have a bounded storage capacity, that is, they can only remember, at any moment, a bounded number of symbols. As before this is technically imposed by considering LSTSes with only balanced actions and by bounding the size of facts. If we restrict actions to be balanced, they neither increase nor decrease the number of facts in the system configuration and therefore 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 in the system.

As shown in [19], protocols and relevant security problems can be modeled by using rewrite rules. In that scenario a set of rewrite rules, or a theory, was proposed for modeling the standard Dolev-Yao intruder [18]. Here, we adapt that theory to model instead an intruder that has a bounded memory, but that still shares many capabilities of the Dolev-Yao intruder, such as the ability to compose, decompose, intercept messages as well as to create fresh values. We will be interested in the same secrecy problem as in [19], namely, in determining whether or not there is a plan which the intruder can use to discover a secret. We also assume that in the initial configuration some agent, A, owns a fact $Q(s')$ with the secret s as the subterm of s'.

Empty facts. For our specifications it will be useful to distinguish the memory storage capacity of the intruder from the memory used in protocol sessions. As in [19], we distinguish some predicate symbols in the alphabet to belong only to

the intruder, among them the predicate symbols M, C, and D. These are used, respectively, when the intruder learns some data, *e.g.*, an encryption key $M(k_e)$, or when he is composing a new message or decomposing a message.

We introduce two types of facts, called *empty facts*, $R(*)$ and $P(*)$ which intuitively denote free memory slots: Empty facts $R(*)$ belong to the intruder, while the empty facts $P(*)$ are used by protocol sessions. As we discuss in more detail later, empty facts $R(*)$ are used by the intruder whenever he learns new data, while empty facts $P(*)$ are used by the participants of the system to create new protocol sessions. As the memory of the intruder is bounded, there is a bound on the number of $R(*)$ facts available. Therefore the intruder might have to manage his memory capacity in order to discover a secret. For instance, whenever the intruder needs to create a nonce or learn some data, he will have to check whether there are empty facts available. Similarly, the number of $P(*)$ facts available in a configuration bounds the number of protocol sessions that can be executed concurrently. So a new protocol session can only be created if there are enough $P(*)$ facts available. The use of $P(*)$ facts implicitly bounds the number of protocol sessions that can be executed concurrently.

6.1. Balanced protocol theories

We modify the rules from [19] that specify the intruder and protocol theories so that only balanced actions are used. In particular, we relax the protocol form imposed in [19], called well-founded theories. In such theories, protocol execution runs are partitioned into three phases: The first phase, called the initialization phase, distributes the shared information among agents, such as the agents' public keys. Only after this phase ends, the second phase called role generation phase starts, where all protocol roles used in the run are assigned to the participants of the system. Finally, after these roles are distributed, the protocol instances run to their completion. Hence, in [19], once protocol sessions start running no new protocol session is created. Here on the other hand, we will relax this assumption and allow protocol sessions to be created and to be "forgotten" while other protocols are running.

Modeling Perfect Encryption. Before we enter into the details of the balanced protocol theories, we introduce some more notation involving encryption taken from [19]. We introduce the alphabet that allows modeling of perfect encryption. The encrypted message represents a "black box" or an opaque message which does not show its contents until it is decrypted with the right key. We consider the following sorts: $cipher$ for ciphertext, *i.e.*, encrypted text, $ekey$ for symmetric

encryption keys, $dkey$ for decryption keys, $nonce$ for nonces, and a sort msg for any type of message. Here we use order-sorted alphabet and have msg as a super-sort and it is the type of the messages exchanged by the participants of the protocol. The following order relations hold among these sorts:

$$nonce < msg, \;\; cipher < msg, \;\; dkey < msg, \;\; ekey < msg.$$

We also use two functions symbols, the pairing function and the encryption function:

$$\langle \cdot, \cdot \rangle : msg \times msg \rightarrow msg \qquad \text{and} \qquad enc : ekey \times msg \rightarrow cipher.$$

As their names suggest, the pairing function is used to pair two messages and the encryption function is used to encrypt a message using an encryption key. Notice that there is no need for a decryption function, since we use pattern-matching (encryption on the left-hand-side of a rule) to express decryption as in [19]. For example, the following rule specifies that if an agent has the correct key then he can decrypt an encrypted message and learn its contents:

$$KP(k_e, k_d)\ A(k_d)\ A(enc(k_e, t)) \rightarrow KP(k_e, k_d)\ A(k_d)\ A(t).$$

The fact $KP(k_e, k_d)$ specifies that k_e and k_d are a pair of encryption and decryption keys. Notice that the rule above is only applicable if the agent A has the right decomposition key, k_d. Otherwise, the rule is not applicable.

Besides the predicate KP, we will use the following predicates to model perfect encryption:

Predicates:

$GoodGuy(ekey, dkey)$:	keys belonging to an honest participant
$BadKey(ekey, dkey)$:	compromised keys known to the intruder
$KP(ekey, dkey)$:	encryption key pair
$AnnK(ekey)$:	published public key

These predicates are basically the same as used in [19]. Keys that belong to an honest participant are contained in $GoodGuy$ facts, while compromised keys are contained in $BadKey$ facts. The $AnnK$ predicate is used to specify public keys that have been published.

For simplicity we will sometimes use $\langle t_1, \ldots, t_{n-1}, t_n \rangle$ for multiple pairing to denote $\langle t_1, \langle \ldots, \langle t_{n-1}, t_n \rangle \rangle \ldots \rangle$. Also, notice that, as in [19], with the use of the pairing function and the encryption function a protocol message is always represented by a single term of the sort msg.

Balanced Role Theories. We now introduce some auxiliary definitions that are going to be used to specify the restrictions on the balanced role theories. These definitions are basically the same as in [19], but adapted to our setting, where all rules are balanced.

Definition 6.1. Let $\mathcal{T}$ be a theory, Q be a predicate and r be a rule, where L is the multiset of facts $F_1, \ldots, F_k$ on the left hand side of r excluding empty facts $R(*)$ and $P(*)$, and R is the multiset of facts $G_1, \ldots, G_n$, possibly with one or more existential quantifiers, on the right hand side of r excluding empty facts $R(*)$ and $P(*)$. A rule in a theory $\mathcal{T}$ *creates* Q facts if some $Q(\vec{t})$ occurs more times in R than in L. A rule in a theory $\mathcal{T}$ *preserves* Q facts if every $Q(\vec{t})$ occurs the same number of times in R and L. A rule in a theory $\mathcal{T}$ *consumes* Q facts if some fact $Q(\vec{t})$ occurs more times in L than in R. A predicate Q in a theory $\mathcal{T}$ is *persistent* if every rule in $\mathcal{T}$ which contains Q either creates or preserves Q facts.

For example, the following rule consumes the fact with predicate A, preserves the B fact, and creates the D fact:

$$A(x)\ B(y) \rightarrow \exists z.B(z)\ D(x).$$

The above definition of the preservation, creation and consumption of facts excludes empty facts, $P(*)$ and $R(*)$, since they do not carry any information. Empty facts only specify an empty slot that can be filled with some non-empty fact.

Definition 6.2. A rule $r = L \rightarrow R$ *enables* a rule $r' = L' \rightarrow R'$ if there exist substitutions σ, σ' such that some fact $P(\vec{t}) \in \sigma R$ created by rule r, is also in $\sigma' L'$. A theory $\mathcal{T}$ *precedes* a theory $\mathcal{S}$ if no rule in $\mathcal{S}$ enables a rule in $\mathcal{T}$.

Intuitively, if a theory $\mathcal{T}$ precedes a theory $\mathcal{S}$, then no facts that appear in the left hand side of rules in $\mathcal{T}$ are created by rules that are in $\mathcal{S}$.

As is usual in protocol security literature, the intruder acts as the network, intercepting and sending messages between the honest participants. We use the public predicate N_S to denote a message that is sent by a participant and that is to be intercepted by the intruder and the public predicate N_R to denote a message that is sent by the intruder to an honest participant. We will explain how the intruder acts as the network later when we introduce the balanced intruder theory.

As in [19] protocols are specified by using role theories containing role states, formally defined below. However, differently from [19], we only allow role theories to contain balanced actions.

Definition 6.3. A theory $\mathcal{A}$ is a *balanced role theory* if there is a finite list of predicates called the *role states* $S_0, S_1, \ldots, S_k$ for some k, and such that all rules in $\mathcal{A}$ are balanced and have one of the following forms:

$$S_0(\ldots)\ P(*)\ W \to_S \exists \vec{z}.S_l(\ldots)\ N_S(\ldots)\ W'$$
$$S_i(\ldots)\ N_R(\ldots)\ W \to_S \exists \vec{z}.S_j(\ldots)\ N_S(\ldots)\ W'$$
$$S_h(\ldots)\ N_R(\ldots)\ W \to_S \exists \vec{z}.S_k(\ldots)\ P(*)\ W'$$

where $l > 0$, $j > i$, $k > h$, W and W' are multisets of facts not containing any role states nor N_S nor N_R facts. We call the first role state, S_0, *initial role state*, and the last role state S_k *final role state*.

Defining roles in this way, ensures that each application of a rule in a balanced role theory $\mathcal{A}$ advances the state forward. The first rule specifies the first step of a protocol session when an initial message is sent in the network, specified by the fact with predicate name N_S. Notice that in order to send this message a $P(*)$ is consumed. If there are no such facts available, then the protocol cannot start. The second rule specifies actions where a participant of the protocol receives a fact from the network, N_R, and sends his response, N_S. In the process, his internal state advances from S_i to S_j, where $j > i$. The third rule specifies the end of the protocol session when the last message is received by a participant and no response is returned. At this point, the participant moves to the last state of the protocol S_k and since no message is sent in the network, a new empty fact $P(*)$ fact is created.

In order to allow the existence of an unbounded number of protocol sessions in a trace, we allow protocol roles to be created at any time with the cost of consuming empty facts $P(*)$. On the other hand, we also allow protocol sessions that have been completed to be forgotten. That is, once a final role state of a session has been reached, it can be deleted, creating new empty facts $P(*)$ in the process. These empty facts can then be used to create new protocol roles starting hence a new protocol session. These theories, called role regeneration theories, are specified in the following definition. Notice that all their actions are also balanced.

Definition 6.4. 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

$$Q_1(\vec{x}_1) \cdots Q_n(\vec{x}_n) P(*) \to Q_1(\vec{x}_1) \cdots Q_n(\vec{x}_n) S_0(\vec{x})$$

where $Q_1(\vec{x}_1) \ldots Q_n(\vec{x}_n)$ is a finite list of persistent facts not involving any role states, and S_0 is the initial role state of one of the theories $\mathcal{A}_1, \ldots, \mathcal{A}_k$, or the form

$$S_k \to P(*)$$

where S_k is the final state of one of the theories $\mathcal{A}_1, \ldots, \mathcal{A}_k$.

Notice that our balanced role theories may contain actions with more than two facts in their pre- and post-conditions. In contrast, the restricted role theories introduced in [19] and used to derive the complexity results in [19] only contain actions with exactly two facts in their pre- and post-conditions (one for the network and the other for the role state). Moreover, although restricted role theories were balanced in [19], role generation theories were not balanced. In well founded theories in [19] one creates all protocol sessions at the beginning of the trace before any protocol session starts executing. Hence, an unbounded number of protocol sessions can run concurrently. The use of un-balanced role generation theories seems to be one source for the undecidability of the secrecy problem. The explicit use of balanced actions in role theories and role regeneration theories is a technical novelty of this paper. It allows us to bound the number of concurrent protocol sessions without bounding the total number of protocol sessions in a trace. The number of protocol roles that can run concurrently is bounded by the number of $P(*)$ facts available, since one needs at least one $P(*)$ fact for every role in a protocol session.

The following definition relaxes well-founded protocols theories in [19] in order to accommodate the creation of roles while protocols are running.

Definition 6.5. A pair $(\mathcal{P}, I)$ is a *semi-founded protocol theory* if I 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 I and the initial and final roles states of the balanced role theories $\mathcal{A}_1, \ldots, \mathcal{A}_n$. 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 semi-founded protocol theory specifies a particular scenario to be model-checked involving some given protocol(s). Besides empty facts, $P(*)$ and $R(*)$, the finite initial set facts contains all the persistent 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.

Remark. In well-founded protocol theories in [19] initialization was achieved by initialization theory $\mathcal{I}$ that preceded role generation and protocol role theories. In that way all the rules from the initialization theory were applied before any other rules. That could also be seen as initial creation of persistent facts that we call initial facts. For simplicity, we follow the assumption in [19, Section 5.1] and prefer the above definition of initialization consisting of a finite number of

persistent facts. However, we are equally able to formulate our theories with a so called balanced sub-theory $\mathcal{I}$ similar to [19]. We can then prove that every derivation in a semi-founded protocol theory can be transformed into a derivation where the rules from the initialization theory are applied first. This is shown in the technical report [26].

6.2. Balanced Intruder Theory

This section introduces a balanced intruder theory following the lines of [19] but for a memory bounded intruder. Similarly to the standard Dolev-Yao intruder [18], he is able to intercept, compose, decompose, decrypt messages whenever he has the corresponding decryption key, as well as create nonces. We assume that the intruder acts as the network, intercepting and sending messages between the honest participants. However, since his memory is bounded, he is constrained by how many free memory slots he has. A free memory slot for the intruder is denoted by empty facts $R(*)$. The intruder will only be able to, for example, learn new data if there are enough $R(*)$ facts available. To use his memory more efficiently, for instance, he might have to forget data already learned, freeing up his memory, before he can learn new data.

Predicates belonging to the Intruder. Besides the empty fact $R(*)$, this paper assumes that the following three unary predicates belong to the intruder:

$D(msg)$:	Decomposable messages known to the intruder.
$M(msg)$:	Information stored in intruder memory.
$C(msg)$:	Composable messages known to the intruder.
$A(msg)$:	Auxiliary fact for deferred decryption.

However, as in [19], more complicated theories where the intruder also distinguishes the sub-types of messages, that is $ekey, dkey$, and $nonce$ can also be specified (see [26]).

Balanced Intruder Theory. Figure 1 contains an example of an intruder theory that uses the predicate names described above and consists of three parts.

The first part called I/O theory has two rules REC and SND. The former specifies the intruder's action of intercepting a message, N_S, sent by an agent, while the latter specifies the action when the intruder sends a message, N_R. Notice the role of the empty facts, $R(*)$ and $P(*)$, in these rules. For instance, when the intruder intercepts a message sent by an honest participants, he consumes one of

I/O Rules:

REC: $N_S(x)\ R(*) \rightarrow D(x)\ P(*)$
SND: $C(x)\ P(*) \rightarrow N_R(x)\ R(*)$

Decomposition Rules:

DCMP: $D(\langle x, y\rangle)\ R(*) \rightarrow D(x)\ D(y)$
LRN: $D(x) \rightarrow M(x)$
DEC: $M(k_d)\ KP(k_e, k_d)\ D(enc(k_e, x))\ R(*)$
$\rightarrow M(k_d)\ KP(k_e, k_d) D(x)\ M(enc(k_e, x))$
LRNA: $D(enc(k_e, x))\ R(*) \rightarrow M(enc(k_e, x))\ A(enc(k_e, x))$
DECA: $M(k_d)\ KP(k_e, k_d)\ A(enc(k_e, x)) \rightarrow M(k_d)\ KP(k_e, k_d)\ D(x)$

Composition Rules:

COMP: $C(x)\ C(y) \rightarrow C(\langle x, y\rangle)\ R(*)$
USE: $M(x)R(*) \rightarrow C(x)M(x)$
ENC: $KP(k_d, k_e)\ M(k_e)\ C(x) \rightarrow KP(k_d, k_e)\ M(k_e)\ C(enc(k_e, x))$
GEN: $R(*) \rightarrow \exists n.M(n)$

Figure 1: Balanced Intruder theory.

Memory maintenance rules:

DELM: $M(x) \rightarrow R(*)$
DELA: $A(x) \rightarrow R(*)$
DELD: $D(x) \rightarrow R(*)$
DELC: $C(x) \rightarrow R(*)$

Figure 2: Memory maintenance theory.

his empty facts, $R(*)$, and creates an empty fact $P(*)$, while the opposite happens when he sends a message.

The second part of the intruder's theory is the decomposition rules, which contains the rules specifying the decomposition of messages as well as the learning of new data by the intruder. For instance, the DCMP rule decomposes a composed message, $D(\langle x, y\rangle)$, into its parts $D(x)$ and $D(y)$, consuming an empty fact $R(*)$ in the process. Thus, if the intruder does not have any $R(*)$ left, that is, no available free memory slots, then the intruder is not able to decompose a message. The rule LRN specifies when a message, $D(x)$, containing some data x is learned by the intruder, denoted by the fact $M(x)$. The rule DECA specifies that the intruder

can decrypt a message whenever he has the right key, while the rule LRNA specifies that when the intruder does not have the key, he can remember a message using the auxiliary predicate A. In that way he can decrypt it later using the rule DECA, if he learns the right key.

The third part contains composition rules, which are symmetric to the decomposition rules. Composition rules specify the basic actions used to compose messages, such as pairing two messages in rule COMP, or using a learned data to compose a message in rule USE, or encrypting a message with a known encryption key in rule ENC, or creating a nonce in rule GEN. Again, notice the role of the empty facts $R(*)$. For instance, when two messages are paired into one, an empty fact $R(*)$ is created, while when creating a nonce an empty fact is consumed. Similarly, in the GEN rule, when the intruder creates a nonce, he consumes an $R(*)$ fact.

As previously mentioned, since our intruder has bounded memory, he might have to manage his memory in a more clever way than the standard Dolev-Yao intruder, which has unbounded memory. In particular, our intruder might need to forget data that he learned, so that he has enough space available in order to learn new information.

The theory that allows the intruder to forget data is called *memory maintenance theory* and is defined below.

Definition 6.6. A theory $\mathcal{E}$ is a *memory maintenance theory* if all of its rules are balanced and their post-conditions consist of the fact $R(*)$, *i.e.*, all the rules have the form $F \to R(*)$, where F is an arbitrary fact belonging to the intruder.

Figure 2 contains the memory maintenance theory for the intruder theory depicted in Figure 1. Since the intruder owns only four predicate names, the memory maintenance theory has four rules. By using them, the intruder can forget any previously learned data, creating new empty facts. These empty facts, on the other hand, can be used by the intruder to learn new data by for instance intercepting another message (REC) or by decomposing some message (DCMP).

Remark. In [19], the notion of normalized derivations was introduced inspired by similar notions in proof theory [42]. In such derivations, decomposition rules always appear before composition rules. It has been argued in [19] that the use of normalized derivations facilitates proof search and normalization was used to prove the decidability of intruder actions. In particular, it is enough to consider

plans where the intruder first decomposes messages before composing new messages. This separation is formalized in [19] by using the different predicates, namely D, C, A and M.

The notion of normalized derivations can be easily adapted to our balanced intruder. However, it might not be always possible to transform a non-normal derivation into a normalized derivation without providing the intruder with more space or memory, *i.e.*with more $R(*)$ facts. The problem is that a permutation of an instance of a COMP rule over an instance of a DCMP rule, might require an extra $R(*)$ fact, as illustrated below:

$$C(a)\ C(b)\ D(c,d) \rightarrow_{COMP} C(a,b)\ R(*)\ D(c,d) \rightarrow_{DCMP} C(a,b)\ D(c)\ D(d).$$

We are unable to switch DCMP and COMP rules unless there is an empty fact in the configuration:

$$C(a)\ C(b)\ D(c,d) \rightarrow_{DCMP} \text{ not enabled } \rightarrow_{COMP} .$$

Pushing COMP rule to the right disabled a rule, since there are no empty facts in the configuration. We, therefore, need an extra memory slot to push the COMP rule to the right, as illustrated below:

$$C(a)\ C(b)\ D(c,d)\ R(*) \rightarrow_{DCMP} C(a)\ C(b)\ D(c)\ D(d) \rightarrow_{COMP} C(a,b)\ R(*)\ D(c)\ D(d).$$

Therefore, if we provide the same number of $R(*)$ facts as the number of decomposition rules in the non-normalized derivation, then one can show that the transformation to a normalized derivation is possible.

Finally, it is worth noting that although normalized derivations are useful for improving proof search, the use of normalized derivations is not necessary for our PSPACE-completeness of the secrecy problem described in Section 7. It would be enough to use a single predicate M to denote the knowledge of the intruder.

6.3. Encoding Known Anomalies with a Bounded Memory Intruder

We can show that many protocol anomalies, such as Lowe's anomaly [35], can also occur when using our bounded memory adversary. We assume that the reader is familiar with such anomalies, see [14, 19, 35, 8, 10]. In this Section, we only demonstrate Lowe's anomaly in detail. However, we have encoded a number of anomalies for other protocols, such as Yahalom [14], Otway-Reese [14, 48], Woo-Lam [14], and Kerberos 5 [8, 10].

Table 2: The size of configurations (m), the number of $R(*)$ facts, the size of configurations modulo intruder (l), and the upper-bound on the size of facts (k) needed to encode protocol runs and known anomalies when using LSTSes with balanced actions. The largest size of facts needed to encode an anomaly is the same as in the corresponding normal run of the protocol. In the cases for the Otway-Rees and the Kerberos 5 protocols, we encode different anomalies, which are identified by the numbering, as follows: [(1)] The type flaw anomaly in [14]; [(2)] The attack 5 in [48]; [(3)] The ticket anomaly and [(4)] the replay anomaly in [8]; [(5)] The PKINIT anomaly also for Kerberos 5 described in [10].

Protocol		Needham Schroeder	Yahalom	Otway Rees	Woo Lam	Kerberos 5	PKINIT[(5)]
Normal	Size of conf. (m)	9	8	8	7	15	18
Anomaly	Size of conf. (m)	19	15	11[(1)], 17[(2)]	8	22[(3)], 20[(4)]	31
	N^o of $R(*)$	7	9	5[(1)], 9[(2)]	2	9[(3)], 4[(4)]	10
	Size mod. intruder (l)	12	6	6[(1)],8[(2)]	6	13[(3)],16[(4)]	21
Upper-bound on size of facts (k)		6	16	26	6	16	28

Table 2 summarizes the number of $P(*)$ and $R(*)$ facts and the upper bound on the size of facts needed to encode normal runs, where no intruder is present, and to encode the anomalies where the bounded memory intruder is present. The *size modulo the intruder* of the configuration is the number of facts in the configuration that do not belong to the intruder. For instance, to realize the Lowe anomaly to the Needham-Schroeder protocol, the intruder requires only seven $R(*)$ facts. Notice that here we only encode standard anomalies described in the literature [8, 14, 48]. This does not mean, however, that there are not any other anomalies that can be carried out by an intruder with less memory, that is, with less $R(*)$ facts.

One can interpret the size of a configuration as an upper bound on how hard is it for a protocol analysis tool to check whether a particular protocol is secure, while the number of $R(*)$ facts can be interpreted as an upper bound on how much memory the intruder needs to carry out an anomaly. The size modulo the intruder can be interpreted as the amount of memory available for protocol sessions. It 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. Therefore, if we bound the size modulo the intruder of configurations, then the amount of $P(*)$

facts is bounded. Furthermore, from Definitions 6.3 and 6.4 one $P(*)$ fact is consumed for every role states created and another $P(*)$ fact is consumed in order to compose the initial message. Therefore, the number of protocol sessions running at the same time is bounded by the number of $P(*)$ facts available, which on the other hand is bounded by the size modulo the intruder of configurations.

Although clearly other factors would need to be taken into account in order to evaluate how secure a protocol is, we believe that the values in Table 2 provides one such factor, namely, the memory required by the intruder to discover a secret. We believe that finding other factors and determining their relations is an interesting research direction. It is, however, out of the scope of this paper and therefore left for future work.

6.4. Lowe anomaly to the Needham-Schroeder protocol

We formalize the well known Lowe anomaly of the Needham-Schroeder protocol [35]. In particular, the intruder uses his memory maintenance theory to handle his memory adequately.

The balanced role theory specifying the Needham-Schroeder protocol is depicted in Figure 3. Predicates A_0, A_1, A_2, B_0, B_1 and B_2 are the role state predi-

Role Regeneration Theory :

ROLA : $GoodGuy(k_e, k_d)P(*) \rightarrow GoodGuy(k_e, k_d)A_0(k_e)$
ROLB : $GoodGuy(k_e, k_d)P(*) \rightarrow GoodGuy(k_e, k_d)B_0(k_e)$
ERASEA : $A_2(k_e, k'_e, x, y) \rightarrow P(*)$
ERASEB : $B_2(k_e, k'_e, x, y) \rightarrow P(*)$

Protocol Theories $\mathcal{A}$ and $\mathcal{B}$:

A1 : $AnnK(k'_e)\ A_0(k_e)P(*)$
$\quad \rightarrow \exists x.A_1(k_e, k'_e, x)\ N_S(enc(k'_e, \langle x, k_e\rangle))\ AnnK(k'_e)$
A2 : $A_1(k_e, k'_e, x)\ N_R(enc(k_e, \langle x, y\rangle)) \rightarrow A_2(k_e, k'_e, x, y)\ N_S(enc(k'_e, y))$
B1 : $B_0(k_e)\ N_R(enc(k_e, \langle x, k'_e\rangle))\ AnnK(k'_e)$
$\quad \rightarrow \exists y.B_1(k_e, k'_e, x, y)\ N_S(enc(k'_e, \langle x, y\rangle))\ AnnK(k'_e)$
B2 : $B_1(k_e k'_e, x, y)\ N_R(enc(k_e, y)) \rightarrow B_2(k_e, k'_e, x, y)\ P(*)$

Figure 3: Balanced semi-founded protocol theory for the Needham-Schroeder Protocol.

cates for initiator and responder roles. First the initiator A (commonly referred to as Alice) sends a message to the responder B (commonly referred to as Bob). The message contains Alice's name, and a freshly chosen nonce, n_a (typically a large random number) encrypted with Bob's public key. Assuming perfect encryption, only somebody with Bob's private key can decrypt that message and learn its content. When Bob receives a message encrypted with his public key, he uses his private key to decrypt it. If it has the expected form (*i.e.*, a name and a nonce), then he replies with a nonce of his own, n_b, along with initiator's (Alice's) nonce, encrypted with Alice's public key. Alice receives the message encrypted with her public key, decrypts it, and if it contains her nonce, Alice replies by returning Bob's nonce, encrypted with his public key. At the end they believe that they are communicating with each other.

The Lowe anomaly has 3 participants to the protocol: Alice, Bob (the beautiful brother) and Charlie (the ugly brother). Alice wants to talk to Bob. However, unfortunately, Bob's key is compromised, so the intruder who knows his decryption key can impersonate Bob, and play an unfair game of passing Alice's messages to Charlie. In particular, the intruder is capable of creating a situation where Alice is convinced that she's talking to Bob while at the same time Charlie is convinced that he's talking to Alice. In reality Alice is talking to Charlie, and Bob receives no messages.

The informal description of Lowe's anomaly is depicted in Figure 4.

This anomaly demonstrates two main points of insecurity for this protocol. First, the nonces n_a and n_c are not secrets between participants who are communicating, Alice and Charlie, because the intruder learns these nonces. The second point regards authentication. The participants in the protocol choose a particular person they want to talk to and at the end of the protocol run they are convinced to have completed a successful conversation with that person. In reality they talk to someone else.

$$A \xrightarrow{\{A, n_a\}_{K_B}} M(B) \xrightarrow{\{A, n_a\}_{K_C}} C$$

$$A \xleftarrow{\{n_a, n_c\}_{K_A}} M(B) \xleftarrow{\{n_a, n_c\}_{K_A}} C$$

$$A \xrightarrow{\{n_c\}_{K_B}} M(B) \xrightarrow{\{n_c\}_{K_C}} C$$

Figure 4: Lowe attack to Needham-Schroeder Protocol

Let us take a closer look at the protocol trace with above anomaly. The initial set of facts contains 9 facts for the protocol participants and 4 facts for the intruder's initial memory. We will call those initial facts W_I.

$$\begin{aligned} W_I = \; & GoodGuy(k_{e1}, k_{d1})\ KP(k_{e1}, k_{d1})\ AnnK(k_{e1}) \\ & BadKey(k_{e2}, k_{d2})\ KP(k_{e2}, k_{d2})\ AnnK(k_{e2}) \\ & GoodGuy(k_{e3}, k_{d3})\ KP(k_{e3}, k_{d3})\ AnnK(k_{e3}) \\ & M(k_{e1})\ M(k_{e2})\ M(k_{d2})\ M(k_{e3}) \end{aligned}$$

A trace representing the anomaly is shown below. Alice starts the protocol by sending the message to Bob, but the intruder intercepts it.

$W_I A_0(k_{e1})\ B_0(k_{e3})\ R(*)R(*)R(*)P(*) \rightarrow_{A1}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ N_S(enc(k_{e2}, \langle n_a, k_{e1}\rangle))\ R(*)R(*)R(*) \rightarrow_{REC}$
$W_I A_1(k_{e1}, k_{e2}, n_a) B_0(k_{e3}) D(enc(k_{e2}, \langle n_a, k_{e1}\rangle))\ R(*)R(*)P(*) \rightarrow$

Intruder has Bob's private key and can therefore decrypt the message. He then encrypts the contents with Charlie's public key, so he sends the message to Charlie pretending to be Alice.

$\rightarrow_{DEC}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ D(\langle n_a, k_{e1}\rangle)\ M(enc(k_{e2}, \langle n_a, k_{e1}\rangle))\ R(*)P(*) \rightarrow_{LRN}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M(\langle n_a, k_{e1}\rangle)\ M(enc(k_{e2}, \langle n_a, k_{e1}\rangle))\ R(*)P(*) \rightarrow_{DEL}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M(\langle n_a, k_{e1}\rangle)\ R(*)\ R(*)P(*) \rightarrow_{USE}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M(\langle n_a, k_{e1}\rangle)\ C(\langle n_a, k_{e1}\rangle)\ R(*)P(*) \rightarrow_{ENC}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M(\langle n_a, k_{e1}\rangle)\ C(enc(k_{e3}, \langle n_a, k_{e1}\rangle))\ R(*)P(*) \rightarrow_{SND}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M(\langle n_a, k_{e1}\rangle)\ N_R(enc(k_{e3}, \langle n_a, k_{e1}\rangle))\ R(*)R(*) \rightarrow_{DEL}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ N_R(enc(k_{e3}, \langle n_a, k_{e1}\rangle)\ R(*)R(*)R(*) \rightarrow$

Additionally the intruder deletes some facts from his memory that he no longer needs using rules from the memory maintenance theory. Charlie receives the message and responds thinking that he is responding to Alice.

$\rightarrow_{B1} W_I A_1(k_{e1}, k_{e2}, n_a)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ N_S(enc(k_{e1}, \langle n_a, n_c\rangle)\ R(*)R(*)R(*) \rightarrow$

The intruder forwards the message to Alice, that is, decomposes the received message and composes the same message.

$\rightarrow_{REC}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ D(enc(k_{e1}, \langle n_a, n_c\rangle)\ R(*)R(*)P(*) \rightarrow_{LRN}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ M(enc(k_{e1}, \langle n_a, n_c\rangle)\ R(*)R(*)P(*) \rightarrow_{USE}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ C(enc(k_{e1}, \langle n_a, n_c\rangle)\ R(*)R(*)P(*) \rightarrow_{SND}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ N_R(enc(k_{e1}, \langle n_a, n_c\rangle)\ R(*)R(*)R(*) \rightarrow$

Alice receives the message, responds (to Charlie) and goes to the final state thinking that she has completed a successful run with Bob.

$\rightarrow_{A2}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ N_S(enc(k_{e2}, n_c))\ R(*)R(*)R(*) \rightarrow_{REC}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ D(enc(k_{e2}, n_c))\ R(*)R(*)P(*) \rightarrow_{DEC}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ M(enc(k_{e2}, n_c))\ D(n_c)\ R(*)P(*) \rightarrow_{DEL}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ R(*)\ D(n_c)\ R(*)P(*) \rightarrow_{LRN}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ R(*)\ M(n_c)\ R(*)P(*) \rightarrow_{USE}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ R(*)\ M(n_c)\ C(n_c)\ P(*) \rightarrow_{ENC}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ R(*)\ M(n_c)\ C(enc(k_{e3}, n_c))\ P(*) \rightarrow_{SND}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ R(*)\ M(n_c)$
$N_R(enc(k_{e3}, n_c))\ R(*) \rightarrow_{(DEL)}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ N_R(enc(k_{e3}, n_c))\ R(*)R(*)R(*) \rightarrow$

Intruder learns Charlie's nonce from Alice's message by decrypting it with the key k_{d2}. He then sends the nonce encrypted with Charlie's public key.

$$\rightarrow_{B2} W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_2(k_{e3}, k_{e1}, n_a, n_c)\ R(*)R(*)R(*)P(*)$$

Charlie receives the message and goes to the final state thinking that he has completed a successful run with Alice.

The anomaly requires a configuration of at least 19 facts in total: 12 $P(*)$ facts for the honest participants (*i.e.*, the size of the configuration modulo the intruder is 12) and 7 $R(*)$ facts for the intruder. The size of facts has to be at least 6.

Remark. Many anomalies are not strictly secrecy problems. For example, they can be authentication anomalies like the Lowe anomaly modeled above. Also, an anomaly can be a run where an intruder discovers a nonce, freshly generated by an honest protocol participant. That is again shown in the Lowe anomaly and is not strictly a secrecy problem. However, nonces are often used as session keys and therefore such anomalies can be reduced to the secrecy problem. When an intruder learns a nonce, *i.e.*, a freshly generated session key that is later used to encrypt the secret, he is also able to learn the secret since he knows the encryption key.

7. Complexity Results for Protocol Theories

In this section we prove a polynomial space complexity result for the secrecy problem of balanced protocol theories with a bounded memory intruder. The *secrecy problem of a protocol theory* is the problem of determining whether or not

an intruder can learn a secret s, *i.e.*of determining whether or not a configuration containing the fact $M(s)$ is reachable from a given initial configuration.

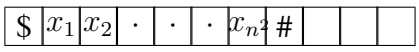
Theorem 7.1. *The secrecy problem with respect to the memory bounded intruder is PSPACE-complete in the size of the balanced semi-founded protocol theory, $(\mathcal{P}, I)$, the size of the balanced intruder theory, $\mathcal{M}$, and the bound, k, on the size of facts.*

PSPACE-hardness. In order to prove the lower bound, we encode a deterministic Turing machine $\mathcal{T}$ that accepts in space n^2 in terms of the secrecy problem.

Without loss of generality, we assume the following:

(a) $\mathcal{T}$ has only one tape, which is one-way infinite to the right. The leftmost cell (numbered by 0) contains the marker \$.

(b) Initially, an *input* string, say $x_1 x_2 \dots x_{n^2}$, is written in cells 1, 2,...,n^2 on the tape. In addition, a special marker # is written in the (n^2+1)-th cell.



(c) The program of $\mathcal{T}$ contains no instruction that could erase either \$ or #. There is no instruction that could move the head of $\mathcal{T}$ either to the right when $\mathcal{T}$ scans symbol #, or to the left when $\mathcal{T}$ scans symbol \$. As a result, $\mathcal{T}$ acts in the space between the two unerased markers.

(d) Finally, $\mathcal{T}$ has only one *accepting* state, and, moreover, all *accepting* configurations in space n are of one and the same form. Moreover, we assume that the accepting state is different from the initial state.

Given an *instantaneous description* (*configuration*) of $\mathcal{T}$ in space n^2 - that $\mathcal{T}$ scans i^{th} cell in state q, where a string $\xi_0\xi_1\xi_2 \dots \xi_i \dots \xi_n\xi_{n+1}$ is written left-justified on the otherwise blank tape, will be represented by the message:

$$\langle \xi_0\xi_1\xi_2 \dots \xi_i \dots \xi_{n^2}\xi_{n^2+1}, q, i\rangle \quad \text{or} \quad \langle \tau, q, i\rangle$$

where τ marks the tape contents. For each machine and an arbitrary initial configuration, encoded by the message $I = \langle \tau_1, q_1, i_1\rangle$, we build a semi-founded protocol theory $(\mathcal{P}_\mathcal{T}, I')$. The initial set of facts is

$$I' = \{Guy(A,k), Guy(B,k), Init(I), Secret(s), 3 \times P(*), 6 \times R(*)\}.$$

The set I' specifies that the agents A and B share the uncompromised key k and contains $\mathcal{T}$'s initial configuration encoded by the message I. Moreover, one needs three $P(*)$ to execute a single protocol session, while the intruder needs at least six empty facts to carry an anomaly: two for storing encrypted messages and the remaining for decomposing and composing messages.

The protocol theory $\mathcal{P}_\mathcal{T}$ is formalized by the following theories for the participants A and B:

Theory for A:

ROLA: $Guy(G,k)Init(I)P(*) \rightarrow_A Guy(G,k)Init(I)A_0(I,k)$

UPDA: $A_0(X,k)P(*) \rightarrow_A A_1(X,k)N_S(\langle update, enc(k,X)\rangle)$

CHKA: $A_1(X,k)N_R(\langle done, enc(k,Y)\rangle) \rightarrow_A A_2(Y,k)N_S(\langle check, enc(k,Y)\rangle)$

RESA: $A_2(X,k)N_R(Res) \rightarrow_A A_3(X,Res,k)P(*)$

ERASEA: $A_3(X,Res,k) \rightarrow_A P(*)$

Theory for B:

ROLB: $Guy(G,k)Secret(s)P(*) \rightarrow Guy(G,k)Secret(s)B_0(k,s)$

UPDB: $B_0(k,s)N_R(\langle update, enc(k,\langle x_0,\ldots,x_{i-1},\xi,x_{i+1},\ldots,x_{n^2+1},q,i\rangle)\rangle)$
$\rightarrow B_1(\langle x_0,\ldots,x_{i-1},\eta,x_{i+1},\ldots,x_{n^2+1},q',i'\rangle,k,s)$
$N_S(\langle done, enc(k,\langle x_0,\ldots,x_{i-1},\eta,x_{i+1},\ldots,x_{n^2+1},q',i'\rangle)\rangle)$

CHKB: $B_1(X,k,s)N_R(\langle check, enc(k,X)\rangle) \rightarrow B_2(X,k,s)N_S(result)$

ERASEB: $B_2(X,k,s) \rightarrow P(*)$

For each instruction γ of the machine $\mathcal{T}$ of the form $q\xi \rightarrow q'\eta D$, denoting *"if in state q looking at symbol ξ, replace it with η, move the tape head one cell in direction D along the tape, and go into state q'"*, is specified by n^2 UPDB rules of B's protocol theory, where $1 \leq i \leq n^2$ is the position of the head of the machine. Hence the reduction is polynomial on n and the number of instructions in $\mathcal{T}$. Both theories for A and for B have the corresponding role generation rules ROLA and ROLB, which create new sessions, as well as 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.

The informal description of the protocol involving A and B is given in Figure 5. The participant A sends a message requesting B to update the encrypted message $\{\langle \tau, q, i\rangle\}_k$ encoding $\mathcal{T}$'s configuration, which includes the state of the machine, head position as well as the contents of the tape. The participant B, who is able to execute instructions of the machine $\mathcal{T}$, deterministically returns the encrypted message $\{\langle \tau', q', i'\rangle\}_k$ encoding the configuration resulting from applying the single instruction to the configuration $\{\langle \tau, q, i\rangle\}_k$. Then the participant A just

$$
\begin{array}{ll}
A \longrightarrow B: & \langle update, \{\langle \tau, q, i\rangle\}_k\rangle \\
B \longrightarrow A: & \langle done, \{\langle \tau', q', i'\rangle\}_k\rangle \\
A \longrightarrow B: & \langle check, \{\langle \tau', q', i'\rangle\}_k\rangle \\
B \longrightarrow A: & result
\end{array}
$$

Figure 5: Normal session for the protocol encoding Turing machines.

bounces this message back to B, so that he can check whether this is the final configuration. If $\{\langle \tau', q', i'\rangle\}_k$ is the accepting configuration then B returns the secret s unencrypted, otherwise if $\{\langle \tau', q', i'\rangle\}_k$ is not the accepting configuration, then he returns the message no also unencrypted.

The informal description of the anomaly carried out by the intruder is depicted in Figure 6. In the first session of the anomaly, the intruder acts as a man-in-the-middle by only overhearing the messages transmitted, that is, he does not modify any of the messages transmitted. In particular, he learns a message $\{X'\}_k$ encoding $\mathcal{T}$'s updated configuration. Notice that since he does not possess the key k, he cannot learn nor modify the message X'. Once the first session is completed, the intruder starts a new session by acting as A and sending a message to B to update the last configuration $\{X\}_k$. Then B returns the new configuration $\{X'\}_k$ encoding the configuration resulting from applying the instruction of $\mathcal{T}$ to the sent configuration X. The intruder then deletes the learned fact $M(\{X\}_k)$ from his memory, freeing his memory to learn the fact $M(\{X'\}_k)$ containing the encoding of the new configuration X'. He then proceeds with the protocol and requests B to check $\{X'\}_k$. If B returns the secret, then X' is encoding the accepting state and the intruder has learned the secret. Otherwise, the intruder starts a new session, again acting as A, but using $\{X'\}_k$ as the initial message. The intruder repeats this process until the secret is revealed, that is, an accepting state is reached. Notice

First Session		**Later Sessions**	
$A \longrightarrow M \longrightarrow B:$	$\langle update, \{\langle \tau, q, i\rangle\}_k\rangle$	$M(A) \longrightarrow B:$	$\langle update, \{\langle \tau, q, i\rangle\}_k\rangle$
$B \longrightarrow M \longrightarrow A:$	$\langle done, \{\langle \tau', q', i'\rangle\}_k\rangle$	$B \longrightarrow M(A):$	$\langle done, \{\langle \tau', q', i'\rangle\}_k\rangle$
$A \longrightarrow M \longrightarrow B:$	$\langle check, \{\langle \tau', q', i'\rangle\}_k\rangle$	$M(A) \longrightarrow B:$	$\langle check, \{\langle \tau', q', i'\rangle\}_k\rangle$
$B \longrightarrow M \longrightarrow A:$	$result$	$B \longrightarrow M(A):$	$result$

Figure 6: Sessions in the anomaly for the protocol encoding Turing machines.

that we need to be careful with the memory of agents. In particular, intruder needs to delete facts from his memory and the participant B needs to delete the fact with the final role state predicate from the previous session before starting a new one.

Lemma 7.2. *Let $(P_{\mathcal{T}}, I')$ be the balanced semi-founded protocol theory encoding the Turing machine $\mathcal{T}$ with the given initial configuration I as described above. Let $\mathcal{M}$ be a balanced two-phase intruder theory with the memory maintenance theory $\mathcal{E}$. A trace obtained from the theory $(P_{\mathcal{T}}, I')$ and $\mathcal{M}$ can lead to a configuration containing the fact $M(s)$, where s is the secret, if and only if the machine $\mathcal{T}$ can reach the accepting state q_f starting from I.*

Proof We now show that the secret is discovered by the intruder M if and only if the machine $\mathcal{T}$ reaches the accepting state.

For the forward direction, assume that there is a sequence of instructions σ that leads the machine $\mathcal{T}$ to the accepting state. Then by induction on the length of σ we can show how to construct a run leading to a state where the secret is revealed. If σ contains just one instruction γ, then the protocol session between agents A and B simulates the application of that instruction reaching the accepting state and exchanging the secret unencrypted, so the intruder can learn the secret simply by intercepting the last protocol message. For the inductive case assume that the sequence of instructions used to reach the accepting state is (γ_1, σ') and that the configuration reached by applying γ_1 is K_2. Moreover, assume that there is an anomaly from the initial configuration containing the fact $M(\{X_2\}_k)$ where X_2 encodes the configuration K_2. We show that there is also an anomaly from a configuration containing the fact $M(\{X_1\}_k)$ encoding the $\mathcal{M}$'s initial configuration K_1. The intruder first sends a request to B to update the message $\{X_1\}_k$. The participant B then uses the action UPDB corresponding to the instruction γ_1, sending the message containing $\{X_2\}_k$. The intruder then deletes the fact $M(\{X_1\}_k)$ and learns the fact $M(\{X_2\}_k)$. When the protocol session is over, the resulting configuration contains the fact $M(\{X_2\}_k)$, for which we can apply the inductive hypothesis ending the proof.

For the reverse direction, we first need the following lemma.

Lemma 7.3. *Let $(P_{\mathcal{T}}, I')$ be the balanced semi-founded protocol theory encoding the deterministic Turing machine $\mathcal{T}$ that accepts in space n^2 and the given initial configuration I of $\mathcal{T}$, as described above. Let $\mathcal{M}$ be a balanced intruder theory. Let $\mathcal{S}$ be an arbitrary configuration reachable from I using $P_{\mathcal{T}}$ and the balanced intruder theory. If the term $\langle \tau, q, i \rangle$ appears in $\mathcal{S}$, then it encodes a configuration reachable from the initial configuration I using $\mathcal{T}$.*

Proof We proceed by induction on the length of a protocol run. For the base case, there are no encrypted messages in I'. For the inductive case, assume that all encrypted terms of the form $\{X\}_k$ appearing in the i^{th} configuration, $\mathcal{S}_i$, in the run encode configurations K_j reachable from I by using $\mathcal{T}$. The only interesting cases are for the rules UPDB in $\mathcal{P}$ and ENC in the intruder theory since they are the only rules that create encrypted messages. The former follows from the definition of $\mathcal{P}$ and the inductive hypothesis: since an application of UPDB simulates one of $\mathcal{T}$'s instructions, γ, and the encrypted term $\{X_j\}_k$ used by it encodes a reachable configuration K_j, the resulting encrypted term created $\{X_{j+1}\}_k$ by this rule encrypts a configuration that is also reachable from I by using the sequence of instructions used to reach the configuration K_j followed by the instruction γ. Now for the latter rule, namely ENC, one can also show by induction on the length of a run that the intruder will never acquire the key k. Therefore the rule ENC is never applicable, that is, the intruder cannot compose terms encrypted with the key k. □

(Returning to the proof of Lemma 7.2). Assume that there is a trace for which the secret is revealed. From the definition of the protocol theory, this is only the case if a message containing the term $\{X\}_k$, where X is the accepting configuration, is received by the participant B. From the previous lemma it must be the case that the accepting configuration X is also reachable from the initial configuration I by using the machine $\mathcal{T}$. □

The upper bound algorithm provided in the proof of Theorem 5.5 for balanced systems in the context of collaborative systems can also be used to determine whether a memory bounded intruder can discover a secret. Following [32], we assume the existence of the function $\mathcal{T}$ that returns, respectively, 1 when given as input a transition that is valid, that is, an instance of an action in the protocol theory or in the intruder theory, and return 0 otherwise. Notice that differently from [32], we do not need other functions that determine whether a configuration contains the fact $M(s)$, as this can be checked in polynomial time. We are now ready to prove the upper bound result.

Theorem 7.4. *There is an algorithm that takes as input:*

1. *a protocol theory $(\mathcal{P}, I)$;*
2. *a balanced intruder theory $\mathcal{M}$;*
3. *an upper bound, k, on the size of facts;*
4. *a program $\mathcal{T}$ that recognizes (in PSPACE) actions of $\mathcal{P}$ and of $\mathcal{M}$;*

which behaves as follows:

(a) *If there is a trace leading from I to a configuration containing the fact $M(s)$, then the algorithm outputs "yes" and schedules a trace; otherwise it returns "no;"*

(b) *It runs in PSPACE with respect to $|\mathcal{P}|$, $|\mathcal{M}|$, $|I|$, $|k|$, and $|\mathcal{T}|$.*

Proof The proof is analogous to the proof of Theorem 5.5. We do not need any critical configurations and moreover all actions in the theories $\mathcal{P}$ and $\mathcal{M}$ are balanced. Therefore, the same algorithm used in the proof of Theorem 5.5 is also applicable here. □

The above PSPACE-completeness result assumes a bound on the size of facts. At first sight such assumption seems unrealistic, as an intruder could in principle construct messages with unbounded depth. In fact, many anomalies, typically the type flaw anomalies, have been discovered, which exploit the fact that messages with the size greater than intended in the protocol are constructed [36] (see also [14]). However, as shown in [4, 25], it is possible to overcome these anomalies by using tagging mechanisms, which are roughly annotations containing the typing information of messages. Thus, by using such mechanisms, agents can reject messages that are of the wrong type. As argued in [4], for such protocols, the assumption on the size of messages is realistic.

8. Related Work

As previously discussed, we build on the framework described in [32, 31]. In particular, here we investigate the use of actions that can create fresh values providing new complexity results for the partial reachability problem. In [6, 7], a temporal logic formalism for modeling organizational processes is introduced. In that framework, one relates the scope of privacy to the specific roles of the agents in the system. We believe that our system can be adapted or extended to accommodate such roles depending on the scenario considered.

In [44], Roscoe formalized the intuition of reusing nonces to model-check protocols where an unbounded number of nonces could be used, by using methods from data independence. We confirm his initial intuition by providing tight complexity results and demonstrating that many protocol anomalies can be specified when using our model that reuses nonces.

Harrison *et al.* present a formal approach to access control [24]. In their proofs, a Turing machine was faithfully encoded 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. 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. However, if actions are not allowed to create fresh values, then they show that the same problem is PSPACE-complete. Furthermore, if actions can delete or insert exactly one fact and in the process check for the presence of other facts and even create nonces, then they show that the problem is NP-complete, but in their proof they implicitly impose a bound on the number of nonces that can be used. In their proofs, the non-commutative nature of the encoding plays an important role.

Our paper is closely related to frameworks based on multiset rewriting systems used to specify and verify security properties of protocols [1, 2, 12, 15, 19, 46]. While here we are concerned with systems where agents are in a *closed room* and collaborate, in those papers, the concern was with systems in an *open room* where an intruder tries to attack the participants of the system by manipulating the transmitted messages. This difference is reflected in the assumptions used by the frameworks. In particular, the security research considers a powerful intruder that has an unbounded memory and that can, for example, accumulate messages at will. On the other hand, we assume here that each agent has a bounded memory, technically imposed by the use of balanced actions.

In this paper, we do not make any assumptions on protocols nor on the format of the exchanged messages, but only that on the memory of the intruder and on the number of parallel protocol sessions.[3] It is possible, however, to recover the decidability of the secrecy problem if further assumptions on the protocol are made even with an unbounded memory intruder and with unbounded number of parallel protocol sessions. For instance, [43] shows that for protocols tagging mechanisms, the secrecy problem is decidable. [16] proposes a general technique to construct safe protocols by using digital signatures linked to protocol sessions. Also [3] proposes a general technique to construct secure protocol by using both digital signature and dynamic tagging mechanisms.

Much work on reachability related problems has been done within the Petri nets (PNs) community, see *e.g.*, [20]. Specifically, we are interested in the *coverability problem* which is closely related to the partial goal reachability problem in LSTSes [31]. To our knowledge, no work that captures exactly the conditions in this paper has yet been proposed. For instance, [20, 38] show that the coverability problem is PSPACE-complete for 1-conservative PNs. While this type of PNs

[3]Even restricting to balanced protocol theories is not an assumption on protocols, as by using empty facts, it is possible to transform a unbalanced protocol theory into a balanced one.

is related to LSTSes with balanced actions, it does not seem possible to provide direct, *faithful* reductions between LSTSes and PNs in this case.

More recently, together with Talcott and Perovic, we have capitalized and extended some of the complexity results in this paper to systems with explicit time [29]. In [41], we also discuss that LSTSes extended with explicit time can be used as the foundations for building an automated assistant to help carry out clinical investigations. Finally, in [40], Nigam identified a fragment of linear authorization logics [21] for which the provability problem is PSPACE-complete. For this result, Nigam used some of the ideas proposed in this paper to handle an unbounded number of nonces.

9. Conclusions and Future Work

This paper extends existing models for collaborative systems with confidentiality policies to include actions that can create fresh values. Then, given a system with balanced actions, we showed that one only needs a polynomial number of constants with respect to the number of facts in the initial configuration and an upper bound on the size of facts to formalize the notion of fresh values. Furthermore, we proved that the weak plan compliance, the plan compliance and the system compliance problems as well as the secrecy problem for systems with balanced actions that can create fresh values are PSPACE-complete. As an application of our results, we showed that a number of anomalies for traditional protocols can be carried out by a bounded memory intruder, whose actions are all balanced.

There are many directions to follow from here, which we are currently working on. Here, we only prove the complexity results for the secrecy problem. We would also like to understand better the impact of our work to existing protocol analysis tools, in particular, our PSPACE upper bound result. Moreover, we are currently working on determining more precise bounds on the memory needed by an intruder to find an attack on a given protocol. Finally, despite of our idealized model, we believe that the numbers appearing in Table 2 provide some measure on the security of protocols. Specifically, the more space required by the intruder to carry an anomaly, the safer one could consider a protocol to be. Clearly other parameters have to be considered. We are currently investigating how to enrich our model in order to include new parameters, such as the number of active sessions running at the same time that are required by the intruder to carry out an attack. In general, we seek to provide further quantitative information on the security of protocols. Some of these parameters appear in existing model checkers, such as Murϕ [17]. We are investigating precise connections to such tools.

Acknowledgments: We thank Elie Bursztein, Iliano Cervesato, Anupam Datta, Ante Derek, George Dinolt, F. Javier Thayer Fabrega, Joshua Guttman, Jonathan Millen, Dale Miller, John Mitchell, Paul Rowe, and Carolyn Talcott for helpful discussions. We also thank the anonymous reviewers for their comments on a previous version of this paper.

Scedrov, Nigam, and Kanovich were partially supported by ONR Grant N00014-07-1-1039, by AFOSR MURI "Collaborative policies and assured information sharing", and by NSF Grants CNS-0524059 and CNS-0830949. This material is based upon work supported by the MURI program under AFOSR Grant No: FA9550-08-1-0352. Nigam was also supported by the Alexander von Humboldt Foundation.

References

[1] Roberto M. Amadio and Denis Lugiez. On the reachability problem in cryptographic protocols. In *CONCUR '00: Proceedings of the 11th International Conference on Concurrency Theory*, pages 380–394, London, UK, 2000. Springer-Verlag.

[2] Roberto M. Amadio, Denis Lugiez, and Vincent Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.*, 290(1):695–740, 2003.

[3] Myrto Arapinis, Stéphanie Delaune, and Steve Kremer. From one session to many: Dynamic tags for security protocols. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR '08, pages 128–142, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Myrto Arapinis and Marie Duflot. Bounding messages for free in security protocols. In *Proceedings of the 27th international conference on Foundations of software technology and theoretical computer science*, FSTTCS'07, pages 376–387, Berlin, Heidelberg, 2007. Springer-Verlag.

[5] Jean-Pierre Banâtre and Daniel Le Métayer. Gamma and the chemical reaction model: ten years after. In *Coordination programming: mechanisms, models and semantics*, pages 3–41. World Scientific Publishing, IC Press, 1996.

[6] Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, pages 184–198, 2006.

[7] Adam Barth, John C. Mitchell, Anupam Datta, and Sharada Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294, 2007.

[8] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, and Christopher Walstad. Formal analysis of Kerberos 5. *Theor. Comput. Sci.*, 367(1-2):57–87, 2006.

[9] CDISC. Study Data Tabulation Model, Section 4.1.2.3. Available at `http://www.cdisc.org/sdtm`.

[10] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Christopher Walstad. Breaking and fixing public-key Kerberos. *Inf. Comput.*, 206(2-4):402–424, 2008.

[11] Iliano Cervesato and Andre Scedrov. Relating state-based and process-based concurrency through linear logic (full-version). *Inf. Comput.*, 207(10):1044–1077, 2009.

[12] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR. *Theor. Comput. Sci.*, 338(1-3):247–274, 2005.

[13] Alonzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[14] John Clark and Jeremy Jacob. A survey of authentication protocol literature: Version 1.0. 1997.

[15] H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 271, Washington, DC, USA, 2003. IEEE Computer Society.

[16] Véronique Cortier, Bogdan Warinschi, and Eugen Zalinescu. Synthesizing secure protocols. In Joachim Biskup and Javier Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 406–421. Springer, 2007.

[17] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In *Proceedings of the 1991 IEEE International Conference on Computer Design on VLSI in Computer & Processors*, ICCD '92, pages 522–525, Washington, DC, USA, 1992. IEEE Computer Society.

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

[19] Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[20] Javier Esparza and Mogens Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.

[21] Deepak Garg, Lujo Bauer, Kevin D. Bowers, Frank Pfenning, and Michael K. Reiter. A linear logic of authorization and knowledge. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 297–312. Springer, 2006.

[22] Gerhard Gentzen. Investigations into logical deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.

[23] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[24] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. On protection in operating systems. In *SOSP '75: Proceedings of the fifth ACM symposium on Operating systems principles*, pages 14–24, New York, NY, USA, 1975. ACM.

[25] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. *J. Comput. Secur.*, 11(2):217–244, March 2003.

[26] Max Kanovich, Tajana Ban Kirigin, Vivek Nigam, and Andre Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. `http://www2.tcs.ifi.lmu.de/~vnigam/docs/tr-bounded.pdf`, August 2010.

[27] Max Kanovich, Tajana Ban Kirigin, Vivek Nigam, and Andre Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. In *FAST*, 2010.

[28] Max Kanovich, Tajana Ban Kirigin, Vivek Nigam, and Andre Scedrov. Progressing collaborative systems. In *FCS-PrivMod*, 2010.

[29] Max Kanovich, Tajana Ban Kirigin, Vivek Nigam, Andre Scedrov, Carolyn Talcott, and Ranko Perovic. A rewriting framework for activities subject to regulations. In *23rd International Conference on Rewriting Techniques and Applications*, 2012.

[30] Max Kanovich, Paul Rowe, and Andre Scedrov. Collaborative planning with privacy. In *CSF '07: Proceedings of the 20th IEEE Computer Security Foundations Symposium*, pages 265–278, Washington, DC, USA, 2007. IEEE Computer Society.

[31] Max Kanovich, Paul Rowe, and Andre Scedrov. Policy compliance in collaborative systems. In *CSF '09: Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium*, pages 218–233, Washington, DC, USA, 2009. IEEE Computer Society.

[32] Max Kanovich, Paul Rowe, and Andre Scedrov. Collaborative planning with confidentiality. *Journal of Automated Reasoning, Special Issue on Computer Security: Foundations and Automated Reasoning*, 2010. To appear. This is an extended version of a previous paper which appeared in CSF'07.

[33] Max Kanovich and Jacqueline Vauzeilles. The classical AI planning problems in the mirror of horn linear logic: semantics, expressibility, complexity. *Mathematical. Structures in Comp. Sci.*, 11:689–716, December 2001.

[34] Peifung E. Lam, John C. Mitchell, and Sharada Sundaram. A formalization of HIPAA for a medical messaging system. In Simone Fischer-Hübner, Costas Lambrinoudakis, and Günther Pernul, editors, *TrustBus*, volume 5695 of *Lecture Notes in Computer Science*, pages 73–85. Springer, 2009.

[35] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS*, pages 147–166, 1996.

[36] Catherine A. Meadows. Analyzing the needham-schroeder public key protocol: A comparison of two approaches. In *In Proc. European Symposium On Research In Computer Security*, pages 351–364. Springer-Verlag, 1996.

[37] Robin Milner. *Communicating and Mobile Systems : The π-calculus*. Cambridge University Press, New York, NY, USA, 1999.

[38] Y.E. Lien N.D. Jones, L.H. Landweber. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

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

[40] Vivek Nigam. On the complexity of linear authorization logics. In *27th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2012.

[41] Vivek Nigam, Tajana Ban Kirigin, Andre Scedrov, Carolyn Talcott, Max Kanovich, and Ranko Perovic. Towards an automated assistant for clinical investigations. In *Second ACM SIGHIT International Health Informatics Symposium*, 2012.

[42] Dag Prawitz. *Natural Deduction*. Almqvist & Wiksell, Uppsala, 1965.

[43] R. Ramanujam and S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. In *Proceedings, Foundations of Software Technology and Theoretical Computer Science (FST TCS 2003), volume 2914 of Lecture Notes in Computer Science*, pages 363–374. Springer, 2003.

[44] A. W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *CSFW*, pages 84–95, 1998.

[45] Paul Rowe. *Policy compliance, confidentiality and complexity in collaborative systems*. PhD thesis, University of Pennsylvania, 2009.

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

[47] W. J. Savitch. Relationship between nondeterministic and deterministic tape classes. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[48] Giulin Wang and Sihan Qing. Two new attacks against Otway-Reese protocol. In *IFIP/SEC2000, Information Security*, pages 137–139, 2000.