Bounded memory Dolev-Yao adversaries in collaborative systems

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

[1] Queen Mary, University of London, UK
mik@dcs.qmul.ac.uk
[2] University of Rijeka, HR
bank@math.uniri.hr
[3] University of Pennsylvania, Philadelphia, USA
{vnigam,scedrov}@math.upenn.edu

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 facts. 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. 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 update values with fresh ones. We investigate the complexity of the decision problem of whether or not an adversary is able to discover secret data. We show that this problem is PSPACE-complete when all actions are balanced and can update values with fresh ones. If we further impose the condition, called progressing, that in a plan any instance of an action can be used at most once, then this new problem is NP-complete when actions are balanced and only a fixed number of updates with fresh values is allowed and it is PSPACE-hard when actions are balanced and any number of updates with fresh values is allowed. Finally, we return to traditional intruder models 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.

1 Introduction

A major concern in any system where agents do not trust each other completely is whether or not the system is secure, that is, whether or not any confidential information or secret of any agent can be leaked to a malicious agent. This paper investigates the complexity of such problem in the context of collaborative system with confidentiality policies [17, 18].

Following [18], 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. This implies that all players inside our system, including adversaries, have a bounded storage capacity, that is, they can only remember at any moment a bounded number of facts. This contrasts with traditional intruder models, which normally include a powerful Dolev-Yao intruder [11] that has an unbounded memory. On the other hand, our adversaries and the standard Dolev-Yao intruder [11] share many capabilities, namely, they are able, within their bounded storage capacity, to compose, decompose, overhear, and intercept messages as well as update values with fresh ones.

This paper shows that the secrecy problem of whether or not an adversary can discover a secret is PSPACE-complete when actions are balanced and can update values with fresh ones. This contrasts with previous results in protocol security literature [12], where it is shown that the same problem is undecidable. However, there they allowed the intruder to have un-balanced actions, or in other words, they assumed that the intruder's memory is not necessarily bounded.

This paper also considers systems that are progressing [16], *i.e.*, systems where plans can contain at most one instance of any action. The notion of progressing is motivated in a similar way as the use of protocol roles in [12]. The idea is that whenever one step of a protocol is performed, one never needs to repeat this step again. We show that, in such progressing systems, the same secrecy problem is NP-complete when actions are balanced and only a fixed number of updates with fresh values is allowed and it is PSPACE-hard when actions are balanced and any number of updates with fresh values is allowed.

Finally, we investigate the consequences of our results in the domain of protocol security. In particular, we demonstrate that when our adversary has *enough* storage capacity, then many protocol anomalies, such as the Lowe anomaly [19] in the Needham-Schroeder public key exchange protocol, can also occur in the presence of a bounded memory intruder. We believe that this is one reason for the successful use in the past years of model checkers to verify protocols. Moreover, we also provide some *quantitative measure* for the security of protocols, namely, the smallest size memory needed by the intruder to carry out the anomalies for several protocols.

This paper is structured as follows: in Section 2 we review the main definitions of local state transition systems used to model collaborative systems. We formalize the notion of fresh values in Section 3, and in Section 4, we summarize the main theoretical results involving the complexity of the different problems considered. We show in Section 5 that many protocol anomalies can also be carried by our bounded memory intruder. Finally, in Sections 6 and 7, we discuss related work and conclude by pointing out some future work.

Some of the results involving the progressing systems also appear in a workshop paper [16] without formal proceedings and the full details are in a technical report [15].

2 Preliminaries

In this section we review the main vocabulary and concepts introduced in [17, 18] and also extend their definitions to accommodate actions that can update values with fresh ones and introduce an adversary.

Local State Transition Systems At the lowest level, we have a first-order signature Σ that consists of a set of sorts 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 (or types). The multi-sorted terms over the signature are expressions formed by applying functions to arguments of the correct sort. Since terms may contain variables, all variables must have assiciated sorts. A *fact* is a ground, atomic predicate over multi-sorted terms. Facts have the form $P(\boldsymbol{t})$ where P is an n-ary predicate symbol and $\boldsymbol{t}$ is an n-tuple of terms, each with its own sort. A *state*, or *configuration* of the system is a finite multiset W of facts. We use both WX and W, X to denote the multiset resulting from the multiset union of W and X.

Definition 1. *The* size of a fact *is the number of term and predicate symbols it contains. We count one for each predicate and function name, and one for each variable or constant symbol. We use $|P|$ to denote the size of a fact P.*

For example, $|P(x, c)| = 3$, and $|P(f(x, n), z)| = 5$. We will normally assume in this paper an upper bound on the size of facts, as in [12, 17, 18].

Following [17, 18], we assume that 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. This separation of the global configuration is done by partitioning the set of predicate symbols in the signature and it will be usually clear from the context. However, differently from [17, 18], we assume that among the agents in the system, there is an adversary M. We also assume the existence of a special constant s in Σ denoting the secret that should not be discovered by the adversary.

As in [17, 18], each agent has a finite set of actions or rules which transform the global configuration. Here, as in [12, 16], we allow agents to have more general actions which can update values with fresh ones (*nonces*). Actions that belong to an agent A have the form: $X_A X_{pub} \rightarrow_A \exists \boldsymbol{t}. Y_A Y_{pub}$. The multisets X_A and Y_A contain facts belonging to the agent A and the multisets X_{pub} and Y_{pub} contain only public facts. Actions work as multiset rewrite rules. All free variables in a rule are treated as universally quantified. $X_A X_{pub}$ are the pre-conditions of the action and $Y_A Y_{pub}$ are the post-conditions of the action. By applying the action for a ground substitution (σ), the pre-condition applied to this substitution ($X_A \sigma X_{pub} \sigma$) is replaced with the post-conditions applied to the same substitution ($Y_A \sigma Y_{pub} \sigma$). In this process, the existentially quantified variables ($\boldsymbol{t}$) appearing in the post-condition are replaced by fresh variables (also known as eigenvariables). The rest of the configuration remains untouched. Thus, we can apply the action $P_A(x), Q_{pub}(y) \rightarrow_A \exists z. R_A(x, z), Q_{pub}(y)$ to the global configuration $V, P_A(t), Q_{pub}(s)$ to get the global configuration $V, R_A(t, c), Q_{pub}(s)$, where the constant c is fresh.

Definition 2. *A* local state transition system *(LSTS) T is a tuple $\langle \Sigma, I, M, R_T, s \rangle$, where Σ is the signature of the language, I is a set of agents, $M \in I$ is the adversary, R_T is the set of actions owned the agents in I, and s is the secret.*

We classify a rule as *balanced* if the number of facts in its precondition is the same as the number of facts in its postcondition. As discussed in [18], if we restrict actions to be balanced, then the size of the configurations in a run remains the same as in the initial configuration. Since we assume facts to have a bounded size, the use of balanced actions imposes a bound on the storage capacity of the agents in the system.

We use the notation $W \rhd_T U$ to mean that there is an action in 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, but only if a configuration contains some particular facts. Therefore we use the notion of partial goals. We write $W \rightsquigarrow_T Z$ to mean that $W \rhd_T ZU$ for some multiset of facts U. For example with the action $r : X \to_A Y$, 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 $W \rightsquigarrow_T^* Z$. 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. A plan is *progressing* [16] if and only if any instance of any action is used at most once in the plan.

In order to achieve a final goal, it is often necessary for an agent to share some private knowledge with some another agent. However, although agents might be willing to share some private information with some agents, they might not be willing to do the same with other agents. For example, a patient might be willing to share his medical history with his doctor, but not with all agents, such as the doctor's secretary. One is, therefore, interested in determining if a system complies with some *confidentiality policies*, such as a patient's medical history should not be publicly available. We call a *critical configurations* any configuration that conflicts with some given confidentiality policies, and we classify any plan that does not reach any critical configuration as *compliant*. A confidentiality policy is determined by a set of critical configurations.

This paper assumes that one can, as in [18], determine in polynomial space with respect to the size of a configuration whether a configuration is a critical. In this paper, we also 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 $\{Paul(n_1, hist), Sec(n_1, hist), Sec(n_1, paul)\}$. This configuration is critical because the secretary knows Paul's medical history, $hist$, since she knows his idendity number, denoted by the nonce n_1, and the medical history 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 [17, 18] several notions of plan compliances were proposed. Here, we consider only the weakest one, called weak plan compliance. This paper makes the additional assumption that initial and the goal configurations are closed under renaming of nonces.

- (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 which leads from W to Z?

Regarding protocol security, we will be interested in the following two different problems. The first problem, called the *secrecy problem*, is basically an instantiation of the weak plan compliance problem with no critical configurations: Is there a plan from the initial configuration to a configuration where the adversary M owns the fact $M(s)$ where s is a secret originally owned by another participant? It is interesting to note that this problem can also be seen as a kind of 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 fact $M(s)$ where s is a secret originally owned by another participant? We also consider another problem, called *secrecy problem for progressing plans*, which is the same as the secrecy problem but when restricted to progressing plans only. Since all players in the system have the same capabilities, if we assume that the system is progressing, then so is the adversary, that is, the adversary is also not allowed to repeat an instance of any action. The progressing notion has a similar effect as the use of role predicates in [12]. While the progressing condition naturally appears in the specification of security of protocols, note that here we differ from [12] because we use only balanced actions. In particular, there, the intruder can copy facts, *i.e.*, the intruder's memory is unbounded.

3 Formalizing Freshness for LSTSes with Balanced Actions

A fresh value can be seen as a new value that has not occurred anywhere in the system yet. For example, when an artist creates his original work of art, it is different to any other song, sculpture or picture developed in human history. However, a fresh value can also be seen as any value that does not belong to any agent of the system in a particular configuration or at a particular moment or period of time. Under the latter interpretation, even values that appeared before in a plan, but that do not appear in a configuration anymore, can be considered fresh. For example, consider the scenario where customers are waiting for a counter. Whenever a new customer arrives, he picks a number and waits until his number is called on. 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 waits 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 be assigned to another customer. This idea of freshness also appears in the execution of protocols. At some moment in a protocol run an agent might need to update a value with a fresh one, or *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, that is, where a third party or intruder cannot overhear messages transmitted.

We can 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. This is formalized by the following theorem:

Theorem 1. *Given an LSTS with balanced actions that can update 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 with respect to the number of facts in W and an upper bound on the size of facts.*

The proof of Theorem 1 relies on the observation that from the perspective of an insider 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 signature:

$$\{A(t_1, n_1), B(n_2, n_1), C(n_3, t_2)\} \quad \text{and} \quad \{A(t_1, n_4), B(n_5, n_4), C(n_6, t_2)\}$$

Since these configurations only differ on 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 [8], we say that these configurations above are α-equivalent.

Formally, 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 nonces appearing in one configuration to the set of all nonces appearing in the other configuration, such that the set $\mathcal{S}_1\sigma = \mathcal{S}_2$. That is, when we apply the complete function derived from σ that maps the nonces from $\mathcal{S}_1$ to the nonces in $\mathcal{S}_2$ to all facts in $\mathcal{S}_1$, then the resulting set is $\mathcal{S}_2$. For example, the two configurations above are α-equivalent because of the following the 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 [20].

Lemma 1. *Let m be the number of facts in a configuration $\mathcal{S}_1$ and a be an upper bound on the size of facts. Let $\mathcal{N}_{m,a}$ be a fixed set of $2ma$ 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 $\mathcal{N}_{m,a}$. Let r be an action whose instance transforms the configuration S_1 into the configuration S_2. Then there is a configuration S_2' such that: (1) an instance of action r can transform 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' still belongs to $\mathcal{N}_{m,a}$.*

Proof Let r be a balanced action that does not update nonces. Let r's instance used to transform $\mathcal{S}_1$ to $\mathcal{S}_2$ contain the nonces $\boldsymbol{n}$ that are in $\mathcal{S}_1$. Let σ be a bijection between the nonces of $\mathcal{S}_1$ and $\mathcal{S}_1'$. Then an instance of r where the nonces n are replaced by $(n\sigma)$ transforms the configuration $\mathcal{S}_1'$ into $\mathcal{S}_2'$. The configurations $\mathcal{S}_2'$ and $\mathcal{S}_2$ are α-equivalent since these configurations differ only in nonce names which are changed by the bijection σ.

Let r be a balanced action that updates nonces. Suppose that some occurrences of nonces $\boldsymbol{n_1}$ within S_1 are updated with fresh nonces $\boldsymbol{n_2}$ resulting in S_2. Note that other places may still keep some of these *old* nonces $\boldsymbol{n_1}$. Take the corresponding occurrence of say $\boldsymbol{n_1}\sigma$ in S_1' (in accordance with our α-equivalence). Since the number of all places is bounded by ma, we can find enough elements (at most ma in the extreme case where all nonces are supposed to be updated simultaneously) $\boldsymbol{n_2'}$ in $\mathcal{N}_{m,a}$ that do not occur in neither $\mathcal{S}_1$ nor $\mathcal{S}_1'$. We update the particular occurrence in question with $\boldsymbol{n_2'}$, resulting in the desired S_2'. Moreover, from the assumption that critical configurations are closed under renaming of nonces and that S_2 is not critical, the configuration S_2' is also not critical. □

We are now ready to prove Theorem 1:

Proof (of Theorem 1). The proof is by induction on the length of a plan and it is based on Lemma 1. Let T be a LSTS with balanced actions that can update nonces, m the number of facts in a configuration, and a the bound on size of each fact. Let $\mathcal{N}_{m,a}$ be a fixed set of $2ma$ nonce names. Given a plan P leading from W to a partial goal Z we adjust it so that all nonces updated along the plan P are taken from $\mathcal{N}_{m,a}$.

For the base case when the plan is of the length 0 it is the case that W already contains Z. Since W is the initial configuration it does not contain any nonces.

Assume that any plan of length n can be transformed in a plan that uses the fixed number of nonces. 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 $Z_1 \to_r ZU$. By induction hypothesis along $W \to_T^* Z_1$, we only have nonces form the set $\mathcal{N}_{m,a}$. We can then apply Lemma 1 to the configuration Z_1 and conclude that all nonces in ZU belong to $\mathcal{N}_{m,a}$. Therefore all nonces updated along the plan P are taken from $\mathcal{N}_{m,a}$. $\Box$

In principle, with the use of nonces, exponential plans can involve an exponential number of nonces. However, Theorem 1 allows one to circumvent this problem.

Corollary 1. *For LSTSes with balanced actions that can update nonces, we only need to consider the planning problem with a polynomial number of* fresh *nonces, 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.*

4 Complexity Results

The following result improves the result in [18, Theorem 6.1] since in their encoding they allowed any type of balanced actions. Here, we tighten their lower bound by showing that LSTSes with balanced actions that can modify at most a single fact and in the process check whether a fact is present in the configuration is also PSPACE-hard. The main challenge here is to simulate operations over a non-commutative structure (tape) by using a commutative one (multiset). (For more discussion on this see [16].) Please also note that in this theorem no nonce updates are allowed.

Theorem 2. *Given an LSTS with only actions of the form $ab \to a'b$, the weak plan compliance problem and the secrecy problem are PSPACE-hard.*

The PSPACE upper bound for this problem can be inferred directly from [17].

Proof In order to prove the lower bound, we encode a non-deterministic Turing machine M that accepts in space n within monadic actions, 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.

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 $i=0,1,..,n{+}1$, ξ is a symbol of the tape alphabet of M, and $S_{j,q}$ denotes that *"the j-th cell is scanned by M in state q"*, where $j=0,1,..,n{+}1$, q is a state of M.

Given a *machine configuration* of M in space n - that M scans j-th cell in state q, when a string $\xi_0\xi_1\xi_2..\xi_i..\xi_n\xi_{n+1}$ 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):

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

Second, each instruction γ in 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:

$$\begin{array}{lll} S_{i,q}R_{i,\xi} \to_A F_{i,\gamma}R_{i,\xi}, & F_{i,\gamma}R_{i,\xi} \to_A F_{i,\gamma}H_{i,\gamma}, & F_{i,\gamma}H_{i,\gamma} \to_A G_{i,\gamma}H_{i,\gamma}, \\ G_{i,\gamma}H_{i,\gamma} \to_A G_{i,\gamma}R_{i,\eta}, & G_{i,\gamma}R_{i,\eta} \to_A S_{i_D,q'}R_{i,\eta}, & \end{array} \quad (2)$$

where $i\!=\!0,1,..,n\!+\!1$, $F_{i,\gamma}$, $G_{i,\gamma}$, $H_{i,\gamma}$ 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 $x_1 x_2..x_n$, 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. Namely, because of the specific form of our actions in (2), any configuration reached by using 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 (2) 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 (1), and, hence, represents a configuration of 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 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 $x_1 x_2..x_n$ is accepted by M. □

We turn our attention to the case when actions can update nonces. We show that the weak plan compliance problem and hence also the secrecy problem for LSTSes with balanced actions that can update nonces is in PSPACE. From Theorem 2, we can infer that this problem is indeed PSPACE-complete.

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 previously reached configuration, there is a cycle of actions which could have been avoided. Thus, at worst, a plan must visit each of the $L_T(m,a)$ configurations, where m is the number of facts in the initial configuration and a an upper bound on the size of facts. The following lemma imposes an upper bound on the number of different configurations given an initial finite signature.

Lemma 2. *Given an LSTSes T under a finite signature Σ, then the number of configurations, $L_T(m,a)$, that are pairwise not α-equivalent and whose number of facts (counting repeatitions) is exactly m is such that $L_T(m,a) \leq J^m(D+2ma)^{ma}$, where J and D are, respectively, the number of predicate and the number of constant and function symbols in the initial signature Σ; and a is an upper bound on the size of facts.*

Proof There are m slots for predicate names and at most ma slots for constants and function symbols. Constants can be either constants in the initial signature Σ or nonce names. Following the Theorem 1, we need to consider only $2ma$ nonces. □

Clearly, the upper bound above 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. In particular, we show next that the secrecy problem for LSTSes with balanced actions that can update nonces is in PSPACE.

Although the secrecy problem is stated as a decision problem, 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, the number of actions in the plan may already be exponential in the size of the inputs, precluding PSPACE membership of plan generation. For this reason we follow [18] 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 3. *An algorithm is said to* schedule *a plan 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 [18], we assume that when given an 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.

Theorem 3. *The weak problem compliance problem and the secrecy problem for LSTSes with balanced actions that can update nonces are in PSPACE.*

Proof Assume as inputs an initial configuration W containing m facts, an upper bound, a, on the size of facts, programs $\mathcal{G}, \mathcal{C}$, and $\mathcal{T}$, as described above, and a natural number $0 \leq i \leq L_T(m, a)$.

We modify the algorithm proposed in [18] in order to accommodate the updating of nonces. The algorithm must return "yes" whenever there is compliant plan from the initial configuration W to a goal configuration. In order to do so, we construct an algorithm that searches non-deterministically whether such configuration *is reachable*, that is, a configuration S such that $\mathcal{G}(S) = 1$. Then we apply Savitch's Theorem 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 a transition 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. From Lemma 2 the goal configuration is reached if and only if it is reached in $L_T(m, a)$ steps. We use a global counter, called step-counter, to keep track of the number of actions used in a partial plan constructed by this algorithm.

In order to accommodate nonce update, we need a way to enforce that whenever an action updates nonces, these are considered fresh. This is done, as in the proof of Theorem 1, by replacing the relevant nonce occurrence(s) with nonces from a fixed set of nonce names so that they are different from any of the nonces in the enabling configuration.

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

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

Therefore, one only needs polynomial space to store the values in the step-counter.

Following the Theorem 1 there are at most polynomialy many nonces updated in any run, namely at most $2ma$. Hence nonces can also be stored in polynomial space.

Table 1. Summary of the complexity results for the secrecy problem. We mark the new results appearing in this paper with a $\star$.

Secrecy Problem		Progressing	Not necessarily progressing
Balanced Actions	Bounded N^o of Nonces	NP-complete*	PSPACE-complete [18]
	Unbounded N^o of Nonces	PSPACE-hard*	PSPACE-complete*
Unbalanced Actions		Undecidable [15]	Undecidable [17]

We must also be careful to check that any configuration, W_t, can also 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. $\Box$

Our PSPACE-complete result contrast with results in [12], where the secrecy problem is shown to be undecidable. Although they also impose an upper bound on the size of facts, they did not restrict the actions of their systems to be balanced. Therefore, it is possible for their intruder to remember an unbounded number of facts, while the memory of all our agents is bounded. Moreover, for their DEXP result, they impose a bound on the number of nonces that can be updated, whereas we do not impose such a bound.

We now move our attention to the secrecy problem for progressing plans.

Theorem 4. *Given an LSTS with only balanced actions, then the secrecy problem for progressing plans is NP-complete when only a bounded number of instances of actions that can update nonces is allowed in a plan.*

Proof The proof for the lower bound is proved by encoding the 3 SAT problem and the NP upper bound by showing that one can check in polynomial time whether any run solves the secrecy problem. The complete proof, which is similar to previous work [2, 7], can be found in [15]. $\Box$

Theorem 5. *Given an LSTS with actions of the form $ab \rightarrow a'b$ or of the form $ab \rightarrow \exists \boldsymbol{t}.a'(t)b$, then the weak plan compliance problem and the secrecy problem for progressing plans are PSPACE-hard.*

Proof (Sketch) The proof goes in a similar fashion as the lower bound proof of Theorem 2. However, we cannot use the same encoding appearing in (2). Since only one instance of any rule in the LSTS can be used, we would only be allowed to encode runs that use an action of M at most once. In order to overcome this problem, here, instead of using propositional rules, we use $6(n+2)$ first-order actions of the form:

$$\begin{array}{ll} S_{i,q}(t)R_{i,\xi} \rightarrow_A \exists t_n.F_{i,\gamma}(t_n)R_{i,\xi}, & F_{i,\gamma}(t)R_{i,\xi} \rightarrow_A F_{i,\gamma}(t)H_{i,\gamma}(t), \\ F_{i,\gamma}(t)H_{i,\gamma}(t) \rightarrow_A G_{i,\gamma}(t)H_{i,\gamma}(t), & G_{i,\gamma}(t)H_{i,\gamma}(t) \rightarrow_A G_{i,\gamma}(t)R_{i,\eta}, \\ G_{i,\gamma}(t)R_{i,\eta} \rightarrow_A S_{i_D,q'}(t)R_{i,\eta}, & S_{i,q}(t) \rightarrow_A S_{i,q}. \end{array} \quad (3)$$

where $i=0,1,..,n+1$. The initial configuration contains a fact $S_{i,q}(c)$ with some constant c and the goal configuration is the accepting configuration with a propositional variable $S_{j,q}$ (of arity zero). Intuitively, the first five rules above are used in the same way as before to encode M's actions of the form $S_{i,q}R_{i,\xi} \rightarrow_A S_{i_D,q'}R_{i,\eta}$, but, now, we create a new constant, t_n, everytime we apply the first rule. This allows us to encode runs where the same action of M is used more than once, since, for each use of this action, we use a different instance of the rules in (3). Moreover, since in the accepting configuration one is not interested in the constant t appearing in the variables $S_{i,q}(t)$, we use the last rule in (3) when the accepting configuration is reached. Notice that after this last action is used, no other rule in (3) is applicable. □

The NP-completeness result is related to a similar result in [12], as both rely on notions similar to progressing. As shown in [18], the secrecy problem without the progressing condition is PSPACE-complete even when actions cannot update nonces. Thus the use of progressing systems improves the complexity for this problem. We also have a EXPSPACE upper bound for the secrecy problem when actions can update nonces [15] and we are currently working on tighter bounds.

Table 1 summarizes the main complexity results for the secrecy problem.

5 Protocol theories with a bounded memory intruder

We return to traditional intruder models and discuss that many protocol anomalies, such as Lowe's anomaly [19], can also occur when using our bounded memory adversary. We assume that the reader is familiar with such anomalies, see [9, 12, 19]. The complete details can be found in the technical report [15].

As in [12], we assume that all messages are transmitted by passing first through the intruder, that is, the intruder acts as the network of the system. We use the public predicate names N_S and N_R to denote messages that are, respectively, sent from an agent to the intruder and from the intruder to another agent. On the other hand, the predicates C, D, and M are private to intruder. The first two are used when he is composing and decomposing messages, respectively, while the third predicate is used to denote some data learned by the intruder. Since the memory or space of agents is bounded, it is important to keep track of how many facts they can store. In particular, the public fact $P(*)$ denotes a free memory-slot available to any agent and the private fact $R(*)$ denotes a free memory-slot available only to the intruder. The use of the two distinct facts for free memory-slots helps us to formalize precise upper-bounds on the space needed by the intruder to realize an anomaly, see [15]. There, we also prove that the secrecy problem is PSPACE-hard when using intruder models, similar to those in [12], but that contain only balanced actions.

We use balanced actions to model the intruder's actions. In particular, our bounded memory Dolev-Yao intruder is also two-phased [12], that is, he first decomposes messages that are intercepted in the network and only then he starts composing new messages. For example, the following rules belong to the intruder:

REC : $N_S(x)R(*) \rightarrow D(x)P(*)$ SND : $C(x)P(*) \rightarrow N_R(x)R(*)$
DCMP : $D(\langle x, y\rangle)R(*) \rightarrow D(x)D(y)$ COMP : $C(x)C(y) \rightarrow C(\langle x, y\rangle)R(*)$
USE : $M(x)R(*) \rightarrow C(x)M(x)$ LRN : $D(x) \rightarrow M(x)$
GEN : $R(*) \rightarrow \exists n.M(n)$

The rules REC and SND specify, respectively, the intruder's actions of intercepting a message from and sending a message to the network. The rules DCMP and COMP specify the intruder's actions of decomposing and composing messages. The rules USE

Table 2. Table containing the total number of facts, the number of $R(*)$ facts, and the largest size of facts 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 [9]; [(2)] The attack 5 in [24]; [(3)] The ticket anomaly and [(4)] the replay anomaly in [5]; [(5)] The PKINIT anomaly also for Kerberos 5 described in [6].

Protocol		Needham Schroeder	Yahalom	Otway Rees	Woo Lam	Kerberos 5	PKINIT[(5)]
Normal	N° of facts	9	8	8	7	15	18
	Size of facts	6	16	26	6	16	28
Anomaly	N° of facts	19	15	11[(1)], 17[(2)]	8	22[(3)], 20[(4)]	31
	N° of $R(*)$	7	9	5[(1)], 9[(2)]	2	9[(3)], 4[(4)]	10

and LRN specify the intruder's actions of using a known data to compose a message and learn some data from an intercepted message. Finally, the rule GEN specifies that the intruder can update fresh values. Notice the role of the facts $P(*)$ and $R(*)$ in the rules. For instance, in the REC rule, when the intruder intercepts a message from the network, one of the the intruder's free memory slots, $R(*)$, is consumed and a free memory slot, $P(*)$, belonging to the other agents is created. The intruder is not allowed to intercept a new network fact if he does not have any free memory slot left.

Therefore, differently from [12] where the intruder had only *persistent* facts, the intruder here might have to forget data. That is, he has actions that replace some fact owned by him by the empty fact $R(*)$, allowing hence the adversary to store eventually new information. For instance, the following rule specifies the intruder's action of forgeting a data known to the intruder: $M(x) \to R(*)$. The complete set of rules for the adversary, including rules involving encryption and decryption, is given in [15].

Regarding protocol anomalies, the main observation is that when the adversary has enough $R(*)$ facts, then anomalies can also occur using adversaries with bounded memory. We believe that this is one reason for the successful use in the past years of model checkers to verify protocols. In the technical report [15], we show that many anomalies can be realized using our bounded memory intruder. 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. We specify protocols using rules that handle encryption and decryption, as in [12]. For instance, to realize the Lowe anomaly to the Needham-Schroeder protocol, one needs a bit more than the twice the number of empty facts as in its normal run, and the intruder requires only seven $R(*)$ facts.[1]

Since all players in our system have a bounded memory, the role generation phase in well-founded theories [12] necessarily yields a bounded number of protocols roles in our system, using here the terminology from [12]. This is because in such theories

[1] Notice that here we only encode standard anomalies described in the literature [5, 9, 24]. 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.

all protocol roles that are used in a run are created at the beginning. Since the size of configurations when using balanced actions is bounded, the number of roles that can be created is also bounded. Thus, under well founded theories, our PSPACE upper bound result (Theorem 3) reduces to the NP upper bound from [12, Theorem 3]. We therefore do not use well-founded theories, but rather allow protocol roles to be created not necessarily at the beginning of the run, but also after a protocol run is finished. Once a protocol session is finished it can be deleted, creating a free memory slot to be (possibly) used to create new protocol roles. Existing protocol analysis tools seem to proceed in a similar fashion.

6 Related Work

As previously discussed, we build on the framework described in [18, 17]. In particular, here we investigate the use of actions that can update values with nonces, providing new complexity results for the partial reachability problem. In [3, 4], a temporal logic formalism for modeling organizational processes is introduced. In their framework, one relates the scope of privacy to the specific roles of agents in the system. We believe that our system can be adapted or extended to accommodate such roles depending on the scenario considered.

In [22], Roscoe formalized the intuition of reusing nonces to model-check protocols where an unbounded number of nonces could be created, 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 [14]. In their proofs, they faithfully encode a Turing machine in their system. However, in contrast to our encoding, they use a non-commutative matrix to encode the sequential, non-commutative tape of a Turing machine. 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 update values with fresh ones, then they show that the same problem is PSPACE-complete. Furthermore, if actions can delete or insert exactly one fact and in the process one can also check for the presence of other facts and even update values with nonces, then they show the problem is NP-complete, but in their proof they implicitly impose a bound on the number of nonces that can be created. In their proofs, the non-commutative nature of their 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, 7, 10, 12, 23]. 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.

Much work on reachability related problems has been done within the Petri nets (PNs) community, see *e.g.*, [13]. Specifically, we are interested in the *coverability problem* which is closely related to the partial goal reachability problem in LSTSes [17].

To our knowledge, no work that captures exactly the conditions in this paper has yet been proposed. For instance, [13, 21] show that the coverability problem is PSPACE-complete for 1-conservative PNs. While this type of PNs is related to LSTSes with balanced actions, it does not seem possible to provide direct, *faithful* reductions between LSTSes and PNs in this case.

7 Conclusions and Future Work

This paper extended existing models for collaborative systems with confidentiality policies to include actions that can update values with fresh ones. 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 and system compliance problems for systems with balanced actions that can update values with fresh ones are PSPACE-complete. We also proved that for systems with balanced actions, the secrecy problem for progressing plans is NP-complete when only a bounded number of nonces can be created, and it is PSPACE-hard when an unbounded number of nonces can be created. Finally, we showed that a number of anomalies for traditional protocols can be carried by a bounded memory intruder, whose actions are all balanced.

There are many directions to follow from here, for which we are currently working on. Here, we only prove the complexity results for the secrecy problem. We are searching for complexity bounds for the weak plan compliance and other policy compliances problems proposed in [17]. 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. We are investigating the consequences of increasing the expressiveness of the language by allowing actions to have constraints, such as arithmetic constraints. 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. 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 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. 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.

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.

References

1. R. M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In *CONCUR '00*, pages 380–394, 2000. Springer-Verlag.

2. R. M. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theor. Comput. Sci.*, 290(1):695–740, 2003.
3. A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, 2006.
4. A. Barth, J. C. Mitchell, A. Datta, and S. Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294, 2007.
5. F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad. Formal analysis of kerberos 5. *Theor. Comput. Sci.*, 367(1-2):57–87, 2006.
6. I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key kerberos. *Inf. Comput.*, 206(2-4):402–424, 2008.
7. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. *Theor. Comput. Sci.*, 338(1-3):247–274, 2005.
8. A. Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.
9. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. 1997. www.cs.york.ac.uk/~jac/papers/drareview.ps.gz
10. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *LICS '03*, page 271, 2003. IEEE Computer Society.
11. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
12. N. A. Durgin, P. Lincoln, J. C. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.
13. J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.
14. M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. On protection in operating systems. In *SOSP '75*, pages 14–24, New York, NY, USA, 1975. ACM.
15. M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Bounded memory Dolev-Yao adversaries in collaborative systems. ftp://ftp.cis.upenn.edu/pub/papers/scedrov/FAST2010-TR.pdf, 2010.
16. M. Kanovich, T. B. Kirigin, V. Nigam, and A. Scedrov. Progressing collaborative systems. In *FCS-PrivMod*, 2010.
17. M. Kanovich, P. Rowe, and A. Scedrov. Policy compliance in collaborative systems. In *CSF '09*, pages 218–233, , 2009. IEEE Computer Society.
18. M. Kanovich, P. Rowe, and A. Scedrov. Collaborative planning with confidentiality. *Journal of Automated Reasoning*, 2010. To appear. This is an extended version of a previous paper which appeared in CSF'07.
19. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS*, pages 147–166, 1996.
20. R. Milner. *Communicating and Mobile Systems : The π-calculus*. Cambridge University Press, New York, NY, USA, 1999.
21. Y. L. N.D. Jones, L.H. Landweber. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.
22. A. W. Roscoe. Proving security protocols with model checkers by data independence techniques. In *CSFW*, pages 84–95, 1998.
23. M. Rusinowitch and M. Turuani. Protocol insecurity with a finite number of sessions and composed keys is NP-complete. *Theor. Comput. Sci.*, 299(1-3):451–475, 2003.
24. G. Wang and S. Qing. Two new attacks against Otway-Reese protocol. In *IFIP/SEC2000, Information Security*, pages 137–139, 2000.