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

Ludwig-Maximilians-Universität, Munich, Germany

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@ifi.lmu.de (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

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 [23, 24].

Following [24], 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, that is, the maximum number of function and constant symbols a fact can contain, then 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 symbols. This contrasts with traditional intruder models, which normally include a powerful Dolev-Yao intruder [14] that has an unbounded memory. On the other hand, our adversaries and the standard Dolev-Yao intruder [14] 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.

This paper shows 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 [15], where it is shown that the same problem is undecidable even when the size of messages is fixed. However, there they allowed the intruder to have unbalanced actions, or in other words, they assumed that the intruder's memory is not necessarily bounded.

In order to obtain a secret, an adversary might need to perform exponentially many actions. Since actions might create fresh values, there might be an exponential number of fresh constants involved in an anomaly, which in principle precludes PSPACE membership. To cope with this problem, we show in Section 4 how to reuse obsolete constants instead of creating new constant names.

Besides the secrecy problem, this paper also investigates the complexity of the three compliance problems introduced in the context of collaborative systems [24, 23], called *weak plan compliance*, *plan compliance*, and *system compliance*. We show that all three problems are also PSPACE-complete when the size of facts is an input bound and when systems contain only balanced actions that can possibly create fresh values.

Although our initial efforts were in collaborative systems, we realized that our results have important consequences for 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 [27] in the Needham-Schroeder public key exchange protocol [30], 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 in protocol verification. 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 [30, 27], Yahalom [11], Otway-Reese [11, 36], Woo-Lam [11], and Kerberos 5 [6, 7].

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:

- Plans constructed using balanced actions can be exponentially long (Section 3);
- We show that when we bound the size of facts, one needs a set with a few nonce names for systems with balanced actions that can create fresh values. The idea is that instead of creating new names, one reuses names (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 we 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 [30, 27], Yahalom [11], Otway-Reese [11, 36], Woo-Lam [11], and Kerberos 5 [6, 7]. The detailed encoding of the Lowe anomaly for the Needham-Schroeder protocol is shown in Section 6.3, while the encoding of anomalies for the other protocols appear in the Appendix.

- We prove the complexity results for the secrecy problem when bounding the size of messages and using balanced systems specifying protocol theories with a bounded memory intruder (Section 7);

Finally, we end by discussing related work and concluding by pointing out some future work in Sections 8 and 9.

This paper extends the paper [20].

2. Preliminaries

In this section we review the main vocabulary and concepts introduced in [23, 24] and also extend their definitions to accommodate actions that can create fresh values and introduce an adversary.

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 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 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 a ground, atomic predicate over multi-sorted terms. Facts have the form $P(\vec{t})$ where P is an n-ary predicate symbol and $\vec{t}$ is an n-tuple of terms, each with its own sort.

Definition 2.1. 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 constant symbol. We use $|P|$ to denote the size of a fact P.

For example, $|P(b, c)| = 3$ and $|P(f(b, n), z)| = 5$. We will normally assume in this paper an upper bound on the size of facts, as in [15].

A *state*, or *configuration* of the system is a finite multiset W of facts. 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 applied to this substitution ($W\sigma$) is replaced with the multiset W' applied to 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 some configuration is reachable from another configuration using a multiset rewrite system. 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 above form have an important limitation, namely, one cannot specify the creation of *fresh values*. These values are often 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 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 fresh value, 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.

As in [15], we borrow the same notion of freshness from proof theory to specify rules that can create fresh values. In particular, in natural deduction systems [17, 31] the elimination rule for the existential quantifier introduces a fresh value, also called *eigenvariable*. This rule is often written in the following way

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

with the side condition that *the constant c does not appear in any other hypothesis*. The rule above states that if we have proved the formula $\exists x.\phi$ and that 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 hipothesis 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 can specify rewrite rules that can create fresh values. These rules have the form $W \to \exists \vec{z}.W'$ and specify that the existentially quantified variables, $\vec{z}$, are to be replaced by fresh values, that is, by 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 $P(x)\ Q(y) \to_A \exists z.R(x,z)\ Q(y)$ to the global configuration $V\ P(t)\ Q(s)$ to get the global configuration $V\ R(t,c)\ Q(s)$, where the constant c must be fresh.

As we will illustrate later in this Section, 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:

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

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 [8, 21] provides a precise connection between the operational semantics of MSRs containing rules that can possibly create fresh values and linear logic derivations [18].

Applications of MSRs. Multiset rewriting systems have been used in several domains. For instance, it has been shown that a wide range of algorithms [3], Artificial Intelligence problems [25, 24], security protocols [15] as well collaborative systems [24, 21] 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.

Local State Transition Systems. In a collaborative system, agents collaborate to achieve a common goal, but they do not completely trust each other. Therefore, while collaborating, an agent might be willing to share some of his private information to some agents, such as when a patient shares his medical history to a doctor, but not willing to share some other information, such as his bank account PIN number.

In order to specify private and shared information, [23, 24] 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 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 above 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. 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 [23, 24], each agent has a finite set of *actions* or *rules*, which transform the global configuration. Here, as in [15, 21, 8], 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:

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

The multisets W_A and W'_A contain 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 [23, 24] 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 postcondition have to be replaced by *fresh values*.

Rules of the form above impose the restriction that any fresh value created by an agent appears only in facts belonging to the agent 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 in an atomic step by using a single instance of an action, such as the one below:

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

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:

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

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, such as in

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

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 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 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 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 $r : X \to_A Y$, 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 $W \rightsquigarrow_T^* Z$. We also consider configurations which 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 [24] 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 have their own set of tasks. For instance, the patients 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 patients blood sample and send it to the lab technician, who would finally perform the required test. This scenario can be specified as a LSTS. The following rules specify some of the actions of the agents N (nurse) and L (lab technician) from this scenario:

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathit{Nurse}(\text{blank}, \text{blank}, \text{blank})\ \mathit{Pat}(\mathit{name}, \mathit{test})} \\
 & \rightarrow_N\ \mathit{Nurse}(\mathit{name}, \text{blank}, \mathit{test})\ \mathit{Pat}(\mathit{name}, \mathit{test}) \\
\mathit{Nurse}(x, \text{blank}, \text{blood}) & \rightarrow_N\ \exists \mathit{id}.\mathit{Nurse}(x, \mathit{id}, \text{blood}) \\
\mathit{Nurse}(x, \mathit{id}, \text{blood}) & \rightarrow_N\ \mathit{Lab}(\mathit{id}, \text{blood})\ \mathit{Nurse}(x, \mathit{id}, \text{blood}) \\
\mathit{Lab}(\mathit{id}, \text{blood}) & \rightarrow_L\ \mathit{TestResult}(\mathit{id}, \text{result})
\end{array}$$

The predicates Pat, Lab and $\mathit{TestResult}$ 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 only the identification number generated in order to anonymize the patient. 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. This is enforced by the specification above 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.

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 [24], 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 a configuration is also inserted into the configuration. Hence, if an LSTS has only balanced actions, then all configurations in a plan have the same number of facts. The sizes of all configurations 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: $\to_A Q_A(x)$. 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, to create a new fact using a balanced action, one needs to replace it with 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 non-empty facts. For instance, the following balanced action creates a new non-empty fact by consuming an empty fact:

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

That is, this action specifies that a free slot can be filled by the fact $Q_A(x)$. Moreover, if an agent does not need to remember some fact, he could free up a slot by this fact by an empty fact, such as specified by the following rule:

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

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 system into balanced systems. For instance, in the medical example shown above, all actions are balanced, except the action:

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

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

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

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 a patient received his test results from the nurse and therefore no longer requires a test to be carried out.

$$\begin{array}{l} Nurse(name, id, blood), TestResult(id, \text{result}), Pat(name, blood) \\ \qquad \to_N Nurse(name, id, blood)\ Rec(name, \text{result})\ P(*) \end{array}$$

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 an agent 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 use for instance 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 an 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 upper bound on the size of facts was also assumed in previous work [15], while [24, 23] assumed fixed the 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 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 test results with the nurse, but not with all agents, such as the lab technician. One is, therefore, 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. 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 fact 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 [26] 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 [23, 24] and the secrecy problem related to protocol security. This paper makes the additional assumption that initial and the goal 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, 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 which 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 which 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) Is there a plan from the 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 reach any of A_i's undesired states.

The type of compliance, *i.e.*, weak plan, system, or plan compliance, required 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 just interested to known whether there is a compliant plan where the goal of writing the paper before the deadline is achieved. [23] provides other illustrative examples.

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

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'.

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 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 we circumvent this problem by reusing obsolete constants instead of creating new names for fresh values. We show that one only requires a small number of nonces in a plan.

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, $\ldots$ 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, 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, and facts of the form $Clear(x)$, where x is of type $diskp$, to 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:

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

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

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

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

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

Finally, the only action in our system is:

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

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 y, specified by predicate $S(x, z)$, or onto a clear peg.

The Towers of Hanoi puzzle representation with LSTSes above 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 folowing two rules.

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

The first rule replaces the old ticket $T(t)$ by 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 which 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 LSTSse in general, more fresh values may be required. We show in the next section, however, 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 prelude 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

[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.

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 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. We call actions that pick fresh values from a small set of nonces as *guarded nonce generation*. Consequently, in a given planning problem we only need to consider a small number of nonces names, which can be fixed in advance. 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:

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

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 [10], 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 nonces appearing in one configuration to the set of all nonces appearing in the other configuration, such that the set $\mathcal{S}_1\sigma = \mathcal{S}_2$.

The two configurations given 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 [28].

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 transform the given transformation $S_1 \to_r S_2$, which can in principle include nonce creation, into $S_1' \to_{r'} S_2'$ so that the action r' does not create new values, instead chooses nonce names from a fixed set given in advance, 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. We say that r' is an action of *guarded nonce generation*.

Let r be a balanced action that does not create nonces. Let r's instance used to transform $\mathcal{S}_1$ to $\mathcal{S}_2$ contain nonces $\vec{n}$ 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 $(\vec{n}\sigma)$ 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.

The most interesting case is 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 critical configurations are closed under renaming of nonces, if S_2 is not critical, the configuration S'_2 is also not critical. $\square$

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 in 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 $Z_1 \to_r ZU$. By induction hypothesis we can transform the plan $W \to^*_T Z_1$ into a plan $W' \to^*_T Z'_1$, 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 the configuration Z_1 and conclude that there is a configuration $Z'U'$ that is α-equivalent to configuration ZU such that all nonces in the configuration $Z'U'$ belong to $\mathcal{N}_{m,k}$. Therefore, all nonces contained in the transformed plan P', *i.e.* in the plan $W' \to^*_T Z'U'$ are taken from $\mathcal{N}_{m,k}$.

Notice that no critical configuration is reached in this process because we assume that critical configurations are closed under renaming of nonce names. $\square$

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

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: No fresh values	Balanced Actions: Possible nonces	Possibly unbalanced actions and no nonces
Secrecy	PSPACE-complete [24]	PSPACE-complete*	Undecidable [15]
Weak Plan	PSPACE-complete [24]	PSPACE-complete*	Undecidable [23]
System	PSPACE-complete [24]	PSPACE-complete*	EXPSPACE-complete [23]; Undecidable with nonces [15]
Plan	PSPACE-complete [24, 33]	PSPACE-complete*	Undecidable [23]

times. However, every time it is used, it appears fresh in the enabling configuration.

5. Complexity Results

In this Section we discuss complexity results for the different 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 the compliance problems discussed in Section 2.

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 [15, 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 [24, Theorem 6.1] since any type of balanced actions was allowed in that encoding. Here, on the other hand, we allow only *monadic actions*, that is 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 $ab \to a'b$, 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 [24].

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 $ab \to a'b$, 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.

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

(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):

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

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:

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

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 (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 (6), 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, specially, 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 (2) 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 state with c and we consider a state 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, after applying the action $ab \to cd$, one necessarily reaches a critical state. In the encoding of Turing machines above, however, 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 then able to simulate plans that include actions of nonce creation with plans containing α-equivalent configurations such that the whole plan only includes 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 the results in the next sections 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 previously reached configuration, there is a cycle of actions which could have been avoided. Thus, at worst, a plan must visit each of the $L_T(m,k)$ configurations, where m is the number of facts in the initial configuration and k 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 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. $\Box$

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.

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 [24] 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 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 [24], 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. For the secrecy problem, we need to additionally assume the 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 on, in Section 6 we give an example of the intruder theory.

Theorem 5.5. *Given an LSTS T with balanced actions that can create nonces and an intruder theory M, then 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 all the same complexity classSavitch. We first prove that the weak plan compliance problem is in PSPACE. We modify the algorithm proposed in [24] in order to accommodate the creation 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 [35] 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. 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 partial 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:

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

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 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.

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 [24] to accommodate the nonce creation. Again we rely on the fact that NPSPACE, PSPACE, and co-PSPACE are all the same complexity class [35]. 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 which 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, called *semi-critical*. 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 system as well as the semi-critical configurations of any agent.

Definition 5.7. A configuration X is *semi-critical for an agent* A if a configuration Y that is critical for A is reachable 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 [33]. Again we rely on the fact that NPSPACE, PSPACE, and co-PSPACE are all 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 less 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 if $\mathcal{C}(j, Z_t) = 1$ 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 that with Theorem 5.5 to conclude that the plan compliance problem is in PSPACE. $\Box$

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 contrast with results in [15], where the secrecy problem is shown to be undecidable. Although in [15] an upper bound on the size of facts was imposed, the actions were not restricted to be balanced. Therefore, 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 [15], 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 [24, 22] 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 [24, 22] 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 [23] that the complexity of both the weak plan and the plan compliance problems are 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 also allow actions to create fresh values. We show next that when actions are possibly unbalanced and can create fresh values, then also the system compliance problem is undecidable.

Theorem 5.9. *The system compliance problem for general LSTSes with actions that can create values with fresh ones 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 [15].

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.

\$	x_1	x_2	$\cdot$	$\cdot$	$\cdot$	x_n	#				$\cdots$

(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 machine $\mathcal{M}$ we construct an LSTS $T_{\mathcal{M}}$ with actions that 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:

$$Adj(c, \#) \to \exists c'.Adj(c, c')\ Adj(c', \#)\ Cont(c', \#)\,. \tag{3}$$

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 $q_i a_s \to q_j a_t L$ 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:

$$Curr(q_i, c)\ Cont(c, a_s)\ Adj(c', c) \to Curr(q_j, c')\ Cont(c, a_t)\ Adj(c', c). \tag{4}$$

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 $q_i a_j \to q_s a_t R$ 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:

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

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 use the first which creates a new cell in the tape.

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 $T_\mathcal{M}$

$$\begin{gathered} Cont(c_0, \$)\ Cont(c_1, x_1) \ldots\ Cont(c_k, x_k)\ Cont(c_{k+1}, \#) \\ Curr(q, c)\ Adj(c_0, c_1)\ \ldots, Adj(c_k, c_{k+1})\,. \end{gathered} \tag{6}$$

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 (3), (4) and (5), which enforce that any reachable configuration has exactly one occurrence of $Curr(q, c)$. Moreover, any reachable configuration is of the form similar to (6), 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. $\square$

6. Application: Protocol theories with bounded memory intruder

This section enters into the details of whether malicious agents, or intruders, with the same capabilities of the other agents are able to discover some secret information. In particular, we modify the intruder theory in [15] 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 [15], 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 [14]. 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 [15], 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 [15], we distinguish some predicate names in the alphabet to belong only to the intruder, among them the predicate names 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 detail in the next sections, 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 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 is some empty fact 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 [15] that specify the intruder and protocol theories so that only balanced actions are used. In particular, we relax the protocol form imposed in [15], called well-founded theories. In such theories, protocols executions 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 [15], 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 [15]. 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. 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 following 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 [15]. 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 [15]. Keys that belong to the an honest participant are contained in $GoodGuy$ facts, while compromised keys 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 [15], 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 auxilary definitions that are going to be used to specify the restrictions on the balanced role theories. These definitions are basically the same as in [15], 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 $P(\vec{t})$ 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 predicate A, preserves the predicate B, and creates the predicate D:

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

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

Definition 6.2. A rule $r = L \to R$ *enables* a rule $r' = L' \to R'$ 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 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 [15] protocols are specified by using role theories containing role states, formally, defined below. However, differently from [15], 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 of one of the following forms:

$$\begin{array}{l} S_0(\ldots)\, P(*)\, W \rightarrow_S \exists \vec{z}.S_l(\ldots)\, N_S(\ldots)\, W' \\ S_i(\ldots)\, N_R(\ldots)\, W \rightarrow_S \exists \vec{z}.S_j(\ldots)\, N_S(\ldots)\, W' \\ S_h(\ldots)\, N_R(\ldots)\, W \rightarrow_S \exists \vec{z}.S_k(\ldots)\, P(*)\, W' \end{array}$$

where $l > 0$, $j > i$, $k > h$, W and W' are multisets of facts not involving 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 in the network, N_R, and sends his reponse, 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 $P(*)$ fact is created.

In order to allow for the existence of an unbounded number of protocol sessions in a trace, we allow protocol roles to be created at any time with the of cost of consuming empty facts $P(*)$. On the other hand, we also allow protocol sessions that have been completed to be forgotten. That is, once its final role state has been reached, it can be deleted, creating in the process new empty facts $P(*)$. These empty facts can then be used to create new protocol roles starting hence a new protocol session. These theories, called role regeneration theories, are specified in the following definition. Notice that all its 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 for one of theories $\mathcal{A}_1, \ldots, \mathcal{A}_k$, or the form

$$S_k \to P(*)$$

where S_k is the final state for one of 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 postconditions. In constrast, the restricted role theories introduced in [15] and used to derive the complexity results in [15] only contain actions with exactly two facts in their pre and postconditions (one for the network and another for the role state). Moreover, although restricted role theories were balanced, role generation theories were not balanced in [15]. In well founded theories in [15] 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 [15] 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 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 [15] initialization was achieved by initialization theory $\mathcal{I}$ that preceded role generation and protocol role theories. In that way all the rules form 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 [15, 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 [15]. We can than prove that every derivation in a semi-founded protocol theory can be transformed into a derivation where the rules from initialization theory are applied first. We include this alternative definition and the proof of this claim in Appendix A.

6.2. Balanced Intruder Theory

This section introduces a balanced intruder theory following the lines of [15] but for a memory bounded intruder. Similarly as the standard Dolev-Yao intruder [14], he is able to intercept, compose, decompose, decrypt messages whenever he has the 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. 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 intruder owns the following three one arity 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 [15], more complicated theories where the intruder also distinguishes the sub-types of messages, that is $ekey, dkey$, and $nonce$ can also be specified. We provide such a theory in Appendix B.

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

I/O Rules:

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

Decomposition Rules:

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

Composition Rules:

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

Figure 1: Balanced Intruder theory.

Memory maintenance rules:

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

Figure 2: Memory maintenance theory.

B, the reader can also find a more refined theory similar to the one in [15] where the intruder also distinguishes the sub-types of messages. For the remainder of the paper, however, it will be enough to use the simple version depicted in Figure 1.

The first part called I/O theory has two rules REC and SND. The former specifies the intruder's action to intercept a message, N_S, sent by an agent, while the latter specifies when the intruder sends a message, N_R. Notice the role of the empty facts, $R(*)$ and $P(*)$, in these rules. For instance, when he intercepts a message sent by an honest participants, the intruder consumes one of 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 smaller 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 more 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, so that he can decrypt it later if he learns the right key using the rule DECA.

The third part contains composition rules, which are symmetric to the decomposition rules. Composition rules specify the basic actions used to compose message, such as pairing two message 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 ENCS, 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 a $R(*)$ 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. This 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 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 only four rules. By using them, the intruder can forget any previously learned data, creating a new empty fact. This new empty fact, 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 [15], the notion of normalized derivations was introduced. In such derivations, decomposition rules always appear before composition rules. Although such a notion could be adapted to our balanced intruder, it might not be always possible to transform a non-normal derivation into a normalized derivation without providing the intruder with more space, that is, with more $R(*)$ facts. The problem is when we attempt to permute an instance of a COMP rule over an instance of a DCMP rule, one might need an extra $R(*)$ fact, as illustrated below:

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

When we try to switch DCMP and COMP rules, we cannot do that because there might be no empty fact in the configuration:

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

Pushing COMP rule to the right disabled a rule, since an empty fact is no longer there. We, therefore, need an extra memory slot to push the COMP rule to the right, as illustrated below:

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

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.

6.3. Encoding Known Anomalies with a Bounded Memory Intruder

We can show that many protocol anomalies, such as Lowe's anomaly [27], can also occur when using our bounded memory adversary. We assume that the reader is familiar with such anomalies, see [11, 15, 27, 6, 7]. In this Section, we only demonstrate Lowe's anomaly in detail. However, in the Appendix, encoding of anomalies for other protocols, such as Yahalom [11], Otway-Reese [11, 36], Woo-Lam [11], and Kerberos 5 [6, 7] are also shown in detail.

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* 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

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 [11]; (2) The attack 5 in [36]; (3) The ticket anomaly and (4) the replay anomaly in [6]; (5) The PKINIT anomaly also for Kerberos 5 described in [7].

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° 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

we only encode standard anomalies described in the literature [6, 11, 36]. 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. We believe that the values in Table 2 provides us with some quantitative information on how secure protocol are.

6.4. Lowe anomaly to the Needham-Schroeder protocol

We formalize the well known Lowe anomaly of the Needham-Schroeder protocol [27]. In particular, the intruder uses his memory maintenance theory to administer 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 predicates 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

Role Regeneration Theory :

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

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

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

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

Bob's nonce, encrypted with his public key. At the end they believe that they are communicating with each other.

The Lowe anomaly (for the other anomalies see Appendix) 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. The informal description of Lowe's anomaly is depicted in Figure 4.

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

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

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

Figure 4: Lowe attack to Needham-Schroeder Protocol

This anomaly demonstrates two main points of insecurity for this protocol. First, the nonces n_a and n_c are not secret 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.

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.

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

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

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

Additionally the intruder deletes some facts from his memory 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 received to Alice, that is, decomposes the received message and composes the same message.

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

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

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

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 sent 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, and 7 $R(*)$ facts for the intruder. The size of facts has to be at least 6.

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 wheather 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 $x_1 x_2 \ldots x_{n^2}$, 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.

\$	x_1	x_2	$\cdot$	$\cdot$	$\cdot$	x_{n^2}	#				$\cdots$

(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 $\xi_0\xi_1\xi_2 \ldots \xi_i \ldots \xi_n\xi_{n+1}$ is written left-justified on the otherwise blank tape, will be represented by the message:

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

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 uncompromissed 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:

$$\begin{array}{ll}
\text{ROLA:} & Guy(G,k)Init(I)P(*) \rightarrow_A Guy(G,k)Init(I)A_0(I,k) \\
\text{UPDA:} & A_0(X,k)P(*) \rightarrow_A A_1(X,k)N_S(\langle update, enc(k,X)\rangle) \\
\text{CHKA:} & A_1(X,k)N_R(\langle done, enc(k,Y)\rangle) \rightarrow_A A_2(Y,k)N_S(\langle check, enc(k,Y)\rangle) \\
\text{RESA:} & A_2(X,k)N_R(Res) \rightarrow_A A_3(X,Res,k)P(*) \\
\text{ERASEA:} & A_3(X,Res,k) \rightarrow_A P(*)
\end{array}$$

Theory for B:

$$\begin{array}{ll}
\text{ROLB:} & Guy(G,k)Secret(s)P(*) \rightarrow Guy(G,k)Secret(s)B_0(k,s) \\
\text{UPDB:} & B_0(k,s)N_R(\langle update, enc(k, \langle x_0,\ldots,x_{i-1},\xi,x_{i+1},\ldots,x_{n^2+1},q,i\rangle)\rangle) \\
 & \rightarrow B_1(\langle x_0,\ldots,x_{i-1},\eta,x_{i+1},\ldots,x_{n^2+1},q',i'\rangle,k,s) \\
 & N_S(\langle done, enc(k,\langle x_0,\ldots,x_{i-1},\eta,x_{i+1},\ldots,x_{n^2+1},q',i'\rangle)\rangle) \\
\text{CHKB:} & B_1(X,k,s)N_R(\langle check, enc(k,X)\rangle) \rightarrow B_2(X,k,s)N_S(result) \\
\text{ERASEB:} & B_2(X,k,s) \rightarrow P(*)
\end{array}$$

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

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

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

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 $\mathcal{T}$'s to the sent configuration X. The intruder then deletes from his memory the learned fact $M(\{X\}_k)$, 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 request 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 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 final role state predicates of the previous session before starting a new one.

Lemma 7.2. *Let $(P_{\mathcal{T}}, I')$ be the balanced semi-founded protocol theory encoding*

First Session **Later Sessions**

$$\begin{array}{rl@{\qquad}rl} A \longrightarrow M \longrightarrow B: & \langle update, \{\langle \tau, q, i\rangle\}_k\rangle & M(A) \longrightarrow B: & \langle update, \{\langle \tau, q, i\rangle\}_k\rangle \\ B \longrightarrow M \longrightarrow A: & \langle done, \{\langle \tau', q', i'\rangle\}_k\rangle & B \longrightarrow M(A): & \langle done, \{\langle \tau', q', i'\rangle\}_k\rangle \\ A \longrightarrow M \longrightarrow B: & \langle check, \{\langle \tau', q', i'\rangle\}_k\rangle & M(A) \longrightarrow B: & \langle check, \{\langle \tau', q', i'\rangle\}_k\rangle \\ B \longrightarrow M \longrightarrow A: & result & B \longrightarrow M(A): & result \end{array}$$

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

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 thory $\mathcal{E}$. 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 dicovered 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 apllying 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 messsage $\{X_1\}_k$. 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 before. 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 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 new encrypted messages. The former follows from the definition of $\mathcal{P}$ and the inductive hypothesis: since an application 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 show also by induction on the length of 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. $\Box$

(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 $\mathcal{T}$. $\Box$

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 [24], 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 [24], 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 recongnizes (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 similar 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. □

Remarks. The decidability of the secrecy problem when the size of facts, the memory available for protocol theories and the memory of the intruder are bounded can have interesting consequences for protocol security. At the current state of affairs, one is only able to decide whether an intruder can find a secret by providing either a bound on the total number of protocol sessions in a trace [2, 34] or by providing a bound on the total number of nonces created in a trace and a bound on the size of facts [15].

However, the bounds described above do not provide useful information on how secure protocols are. For instance, when no anomaly is found for a given protocol and for some given bounds, one can only make statements of the following form: "the protocol is secure if it is used at most n times" or "the protocol is secure if at most m nonces are created." Unfortunately, such statements do not provide tangible quantitative measures on the security of protocols. It is normally expected that agents establish secure channels using the same protocols an unbounded number of times and creating an unbounded number of nonces. For instance, a bank customer usually checks his online statement, accessing his personal online bank homepage and inserting his online PIN number, an unbounded number times.

On the other hand, when using our approach and when no anomaly is found for a protocol given some bounds on the size of facts, the memory available for protocols and the memory of the intruder, one can extract some tangible quantitative information on how secure the protocols are. The size of facts corresponds to the size of the messages exchanged. As discussed in Section 6, the bound on the memory available for protocol sessions bounds the number of concurrent protocol sessions in a trace. Many e-mail providers, online banking systems and game servers disallow the same user to be logged-in more than once by using, for example, different computers. Hence, the same user cannot participate in two concurrent protocol sessions. Finally, the bound on the memory of the intruder also provides a quantitative information on the power of the intruder. The more memory he has, the more powerful he is. We do not require a bound on the length of the trace.

The quantitative use of the bounds mentioned above is left to future work.

8. Related Work

As previously discussed, we build on the framework described in [24, 23]. In particular, here we investigate the use of actions that can create values with nonces, providing new complexity results for the partial reachability problem. In [4, 5], 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 [32], 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 [19]. 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 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 one can also check for the presence of other facts and even create 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 used. 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, 9, 12, 15, 34]. 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.*, [16]. Specifically, we are interested in the *coverability problem* which is closely related to the partial goal reachability problem in LSTSes [23]. To our knowledge, no work that captures exactly the conditions in this paper has yet been proposed. For instance, [16, 29] 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.

9. Conclusions and Future Work

This paper extended 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. We are investigating the consequences of increasing the expressiveness of the language by allowing actions to have constraints, such as arithmetic constraints, as well as adding explicit time to our model. 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 [13]. 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. 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] 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.

[4] 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.

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

[6] 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.

[7] 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.

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

[9] 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.

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

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

[12] 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.

[13] 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.

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

[15] 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.

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

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

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

[19] 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.

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

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

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

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

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

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

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

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

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

[34] 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.

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

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

Appendix A. Alternative definition of semi-founded protocol theory

Definition Appendix A.1. A theory $\mathcal{S} \subset \mathcal{T}$ is a *bounded sub-theory* of $\mathcal{T}$ if all formulas on the right hand side of the rules R in $\mathcal{S}$ either contain existentials or are persistent in $\mathcal{T}$.

Definition Appendix A.2. A theory $\mathcal{P}$ is a *semi-founded protocol theory* if $\mathcal{P} = \mathcal{I} \uplus \mathcal{R} \uplus \mathcal{A}_1 \uplus \cdots \uplus \mathcal{A}_n$ where $\mathcal{I}$ is a bounded sub-theory (called the *initialization theory*) not involving any role states, $\mathcal{R}$ is a role regeneration theory involving only facts created by $\mathcal{I}$ and the initial and final roles states of $\mathcal{A}_1, \ldots, \mathcal{A}_n$, and $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are bounded role theories, with $\mathcal{I}$ preceding $\mathcal{R}$ and $\mathcal{R}$ preceding $\mathcal{A}_1, \ldots, \mathcal{A}_n$. 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 $\mathcal{A}_j$.

$$\begin{array}{l}
\text{GOODGUY} : P(*)P(*) \rightarrow \exists k_e.k_d.GoodGuy(k_e, k_d)KP(k_e, k_d) \\
\text{BADKEY} : P(*)P(*) \rightarrow \exists k_e.k_d.BadKey(k_e, k_d)KP(k_e, k_d) \\
\text{ANNK} : GoodGuy(k_e, k_d)P(*) \rightarrow AnnK(k_e)GoodGuy(k_e, k_d) \\
\text{ANNKB} : BadKey(k_e, k_d)P(*) \rightarrow AnnK(k_e)BadKey(k_e, k_d)
\end{array}$$

Figure A.7: Initialization theory for the Needham-Schroeder Protocol.

The next proposition shows that semi-restricted protocol form allows derivations in a protocol theory to be broken down into two stages: the initialization stage and the stage in which the rules from the role regeneration theory and the protocol role theories are interleaved to allow an unbounded number of roles. Also, from the point of view of the memory deleting final role states provides some free space for storage of any facts, not just for new initial role predicates.

Lemma Appendix A.3. *In a semi-founded protocol theory $\mathcal{P} = \mathcal{I} \uplus \mathcal{R} \uplus \mathbf{A}$, where $\mathbf{A} = \mathcal{A}_1 \uplus \cdots \uplus \mathcal{A}_p$, for any derivation $S \rhd^* T$ with n participants there exists such a derivation*

$$SP(*)^{3p \cdot n^2} \rightsquigarrow^*_{\mathcal{I}} S', \;\; S' \rightsquigarrow^*_{\mathcal{R} \uplus \mathbf{A}} T.$$

In other words, all rules from $\mathcal{I}$ are applied before any rules from $\mathcal{R}$ and any rules from $\mathbf{A}$.

Proof Since $\mathcal{P}$ is a semi-founded protocol theory, no rules in $\mathcal{R}$ and $\mathbf{A}$ can enable rules in $\mathcal{I}$, therefore all rules from $\mathcal{I}$ can be applied before any rules in $\mathcal{R}$ and $\mathbf{A}$.

Anyway, when the rules from the given derivations are rearranged in the above way, the treatment of memory has to be considered. Initialization rules consume empty facts and create persistent facts, so they do not free any memory slots. Therefore the number of empty facts consumed by initialization rules is the same regardless of the order in which the rules are applied. Since the given derivation $S \rhd^* T$ was possible, the required number of empty slots was available in S or it was created by other rules that consume facts to leave free memory slots. One such rule is the rule that deletes final role state: ERASE : $S_k \to R(*)$.

Each time ERASE rule creates an empty fact, it is there in the configuration, available for another session, *i.e.*for the rule that creates an initial state. Since there are 2 ERASE rules per role theory and the roles are parameterized by key pairs (k_e, k_e'), there are at most $2p \cdot n(n-1)$ opportunities for initialization rules to consume those empty fact (the number of possible combinations of initiator and responder per role theory).

Another rule that leaves empty fact is the rule from bounded role theories; the rule that has the final role state together with an empty fact in the post-condition. In bounded protocol role theories, other rules from role theories do not create empty facts. Therefore we need additional $n(n-1)$ empty facts for these rules; one for each combination of keys (*i.e.*participants) for the session, but only one of them has the final rule with the empty fact. Therefore, in total, we need $3p \cdot n(n-1)$ additional empty facts required the transformation. $\Box$

Appendix B. Typed signature for Protocol and intruder theories

In our analysis, we consider several protocols, some of which require additional data types such as timestamps and certificates, and different types of encryption to the private/public key encryption in the Needham-Schroeder protocol. Figures B.8, B.9 and B.10 show the extended typed alphabet.

Predicates used in the protocol theory will depend of the particular protocol that is represented. For simplicity, with asymmetric encryption we identify the principal with its public key (*i.e.*, we use the public key "k_a" to indicate that A is participating in the protocol and has the public key k_a)

Sorts :

$ekey$:	encryption key (and principal name)
$dkey$:	decryption key
$keys$:	key for symmetric encryption
key :	key for any encryption
$cipher$:	cipher text (encrypted)
$nonce$:	nonces
$msgaux$:	auxiliary type for generic message generation
guy :	participant in the protocol
$time$:	timestamp or lifetime
$cert$:	certificate in PKINIT
msg :	data of any type

Subsorts :

$nonce < msg,\ \ cipher < msg,$
$ekey < key,\ \ dkey < key$
$skey < key,\ \ key < msg$
$msgaux < msg\ \ guy < msg$
$time < msg,\ \ cert < msg$

Functions :

$enc : key \times msg \to cipher$:	encryption
$\langle , \rangle : msg \times msg \to msg$:	pairing

Figure B.8: Types and functions for the protocol theories

Predicates :

$GoodGuy(ekey, dkey)$	: identity of an honest participant with private and public keys
$Guy(guy, key)$	: identity of a participant with symmetric key
$BadKey(ekey, dkey)$	: keys of a dishonest participant
$KP(ekey, dkey)$	: encryption key pair
$AnnK(ekey)$	: published public key
$Server(guy)$	: name of a Server
$ServerKey(guy, key)$	: identity of a Server with symmetric key
$N(cipher)$	: encrypted message on the network (sent or received)
$N_S(cipher)$	: encrypted message (sent)
$N_R(cipher)$	: encrypted message (received)
$A_i, B_i, \ldots$	role state predicates (types change per protocol)
$R(*), B(*)$	: empty facts in intruder's memory
$D(msg)$	: decomposable fact in intruder's memory
$C(msg)$	: fact being composed by intruder in intruder's memory
$A(msg)$	: auxiliary opaque fact in intruder's memory
$M_{ek}(ekey)$	: agent's public key in intruder's memory
$M_{dk}(dkey)$	: agent's private key in intruder's memory
$M_k(key)$	: symmetric key in intruder's memory
$M_n(nonce)$	: nonce in intruder's memory
$M_g(guy)$	: participant's name in intruder's memory
$M_m(msgaux)$	: generic message in intruder's memory
$M_s(msg)$	: intercepted submessage in intruder's memory
$M_t(time)$	: timestamp in intruder's memory
$M_l(time)$	: lifetime in intruder's memory
$M_p(cert)$	: certificate in intruder's memory

Figure B.9: Predicates for the Protocol theories

Predicates in Kerberos 5 Protocol:

$KAS(guy)$: name of a Kerberos Authentication Server
$TGS(guy)$: name of a Ticket Granting Server
$TGSKey(guy, key)$: identity of a TGS with symmetric key
$Auth_C(msg, guy, keys)$: memory predicate for the ticket granting ticket
$Service_C(msg, guy, keys)$: memory predicate for the service ticket
$Valid_K(guy, guy, nonce)$: constraint for validitiy of request to KAS
$Valid_T(guy, guy, nonce)$: constraint for validitiy of request to TGS
$Valid_S(guy, time)$: constraint for validitiy of request to Server
$Clock_C(time)$: constraint for time in Kerberos 5 and PKINIT
$Clock_K(time)$: constraint for time in PKINIT
$DoneMut_C(guy, keys)$: memory predicate for succesful mutual authentication
$Mem_S(guy, keys, time)$: memory predicate for mutual authentication completed

Figure B.10: Predicates specific to the Kerberos Protocols

While in the case of private/public encryption we can identify the participants name with his public key, for protocols that use symmetric encryption, we identify the set of participants owning symmetric keys by using the predicate Guy. For the intruder we use the predicate M_g for storing participants' (guys') names and M_k for storing symmetric keys for encryption/decryption.

In addition to symmetric encryption, we model the encryption with composed keys to allow some type-flaw anomalies, such as the anomaly for the Otway-Reese protocol described in [11]. Such attacks are prevented by typed alphabets such as ours so we need to allow this kind of encryption to represent these attacks by adding the new type $msgaux$.

Finally, there are also protocols that use digital signatuires. We represent them with encryptions with private keys whose public keys are announced and therefore the signature can be checked by "decrypting with public keys." Notice that with the use of subsorts the function enc has been extended to include other types of encryption.

Predicates $Server$, $ServerKey$, KAS, TGS, $TGSKey$ shown in Figure B.10 are related to Servers participating in protocols, including specific Kerberos servers. There are additional predicates related to Kerberos protocol that represent tickets, authentication, clocks and validity constrains: $Auth_C$, $Service_C$, $Valild_K$, $Valild_T$, $Valild_S$. $Clock_C$, $Clock_K$, $DoneMut_C$ and Mem_S.

Other predicates private to the intruder include predicates R and B exclusively denoting empty facts, *i.e.* intruder's available memory. Predicate M_s stores any submessage intruder intercepted, predicate M_t represents timestamps, M_l represents lifetimes, M_p represents certificates in Public key extension of Kerberos PKINIT.

Also notice that all the predicates private to the intruder, *e.g.*, D, C, A and various $M_?$ predicates, are unary predicates. This is because complex messages are built by using the pair, $\langle\cdot\rangle$, and encryption function, enc. Therefore, in order to interact with the other participants, the intruder does not require predicates with greater arity, but only pattern match terms using these functions.

As the Dolev-Yao intruder specified in [15], our bounded memory intruder is still able, provided he has enough memory slots vailable, to intercept messages from the network, send messages onto the network, compose and decompose, and decrypt and encrypt messages with available keys. In addition to these capabilities our intruder is able to use his memory as economically as possible and therefore carry out anomalies using less memory space. This new, more clever intruder, will digest only those messages and parts of the messages that contain data that is useful for the attack.

The balanced intruder theory with rules similar to those in [15] and similar to the intruder theory described in Section 6 in Figure 1 plus the additional rules for new sorts and types of encryption is depicted in Figure B.11. Additional rules that enable the intruder to use his memory more cleverly are depicted in Figure B.13. Finally, his memory maintenance theory is depicted in Figure B.12.

Various LRN rules convert decomposable facts into intruder knowledge, and USE rules convert intruder knowledge into a composable fact. These sets of rules are typed, *i.e.*, USEN reads a nonce from the intruder memory and makes that nonce available for composition of a message.

Symmetric encryption is modeled by encryption and decryption rules, ENCS and DECS, as well as the auxiliary rules LRNAS and DECAS. Encryption with composed keys is represented by the ENCM rule. The rules SIG and DSIG represent signatures by encrypting with a private keys whose public key is announced and by checking the signature "decrypting" with the matching public key.

GENM rule generates a generic message to perform "ticket anomaly" in Kerberos 5 shown in Appendix Appendix G.1. Intruder should be able to generate a generic message of the type $msgaux < msg$ in a separate memory predicate M_m representing a "false ticket". Type $msgaux$ is required to retain storing of different subtypes of messages in separate memory facts. If the msg type was used instead, any term could be stored in the memory fact M_m.

I/O Rules:

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

Decomposition Rules:

DCMP : $D(\langle x, y\rangle)R(*) \to D(x)D(y)$
LRNEK : $D(k_e) \to M_{ek}(k_e)$
LRNDK : $D(k_d) \to M_{dk}(k_d)$
LRNK : $D(k_e) \to M_k(k)$
LRNN : $D(n) \to M_n(n)$
LRNG : $D(G) \to M_g(G)$
LRNT : $D(t) \to M_t(t)$
LRNL : $D(l) \to M_l(L)$
LRNP : $D(x) \to M_p(x)$
LRNM : $D(m) \to M_m(m)$
DEC : $M_{dk}(k_d)KP(k_e, k_d)D(enc(k_e, x))R(*)$
$\to M_{dk}(k_d)KP(k_e, k_d)D(x)M_c(enc(k_e, x))$
LRNA : $D(enc(k_e, x))R(*) \to M_c(enc(k_e, x))A(enc(k_e, x))$
DECA : $M_{dkn}(k_d)KP(k_e, k_d)A(enc(k_e, x)) \to M_{dk}(k_d)KP(k_e, k_d)D(x)$
DECS : $M_k(k)\ D(enc(k, x))\ R(*) \to M_k(k)\ M_c(enc(k, x))\ D(x)$
LRNAS : $D(enc(k, x))R(*) \to M_c(enc(k, x))A(enc(k, x))$
DECAS : $M_k(k)A(enc(k, x)) \to M_k(k)D(x)$
DSIG : $M_{ek}(k_e)KP(k_e, k_d)D(enc(k_d, x))R(*) \to$
$M_{ek}(k_e)KP(k_e, k_d)D(x)M_c(enc(k_d, x))$

Composition Rules:

COMP : $C(x)C(y) \to C(\langle x, y\rangle)R(*)$
USEEK : $M_{ek}(k_e)R(*) \to C(k_e)M_{ek}(k_e)$
USEDK : $M_{dk}(k_d)R(*) \to C(k_d)M_{dk}(k_d)$
USEK : $M_k(k)R(*) \to C(k)M_k(k)$
USEN : $M_n(n)R(*) \to C(n)M_n(n)$
USEC : $M_c(c)R(*) \to C(c)M_c(c)$
USEG : $M_g(c)\ R(*) \to C(c)\ M_g(c)$
USET : $M_t(t)R(*) \to M_t(t)\ C(t)$
USEL : $M_l(L)R(*) \to M_l(L)\ C(L)$
USEM : $M_m(m)R(*) \to M_m(m)\ C(m)$
USEP : $M_p(x)R(*) \to M_p(x)\ C(x)$
ENC : $M_{ek}(k_e)C(x) \to C(enc(k_e, x))M_{ek}(k_e)$
ENCS : $M_k(k)\ C(x) \to M_k(k)\ C(enc(k, x)),$
ENCM : $C(x)C(y) \to M_k(x)C(enc(x, y))$
SIG : $M_{dk}(k_d)C(x) \to M_{dk}(k_d)C(enc(k_d, x))$
GEN : $R(*) \to \exists n.M_n(n)$
GENM : $R(*) \to \exists m.M_m(m)$

Figure B.11: Two-phase Intruder theory.

Memory maintenance rules:

$$
\begin{aligned}
\text{DELEK} &: M_{ek}(x) \rightarrow R(*)\\
\text{DELDK} &: M_{dk}(x) \rightarrow R(*)\\
\text{DELK} &: M_{k}(x) \rightarrow R(*)\\
\text{DELN} &: M_{n}(x) \rightarrow R(*)\\
\text{DELC} &: M_{c}(x) \rightarrow R(*)\\
\text{DELG} &: M_{g}(G) \rightarrow R(*)\\
\text{DELT} &: M_{t}(t) \rightarrow R(*)\\
\text{DELL} &: M_{l}(l) \rightarrow R(*)\\
\text{DELP} &: M_{p}(x) \rightarrow R(*)\\
\text{DELM} &: M_{m}(m) \rightarrow R(*)\\
\text{DELB} &: B(*) \rightarrow R(*)
\end{aligned}
$$

Figure B.12: Memory maintenance theory.

Decomposition Rules:

$$
\begin{aligned}
\text{DM} &: D(x) \rightarrow M_s(x)\\
\text{DELD} &: D(m) \rightarrow B(*)\\
\text{DELAB} &: A(m) \rightarrow B(*)\\
\text{DELMC} &: M_c(m) \rightarrow B(*)\\
\text{DCMPB} &: D(\langle x, y\rangle)\ B(*) \rightarrow D(x)\ D(y)\\
\text{DECB} &: M_{dk}(k_d)\ KP(k_e, k_d)\ D(enc(k_e, x))\ B(*) \rightarrow\\
&\quad M_{dk}(k_d)\ KP(k_e, k_d)\ D(x)\ M_c(enc(k_e, x))\\
\text{DSIGB} &: M_{ek}(k_e) KP(k_e, k_d) D(enc(k_d, x)) B(*) \rightarrow\\
&\quad M_{ek}(k_e) KP(k_e, k_d) D(x) M_c(enc(k_d, x))\\
\text{LRNAB} &: D(enc(k_e, x))\ B(*) \rightarrow M_c(enc(k_e, x))\ A(enc(k_e, x))
\end{aligned}
$$

Composition Rules:

$$
\text{USES} : M_s(*)\ R(*) \rightarrow M_s(m)\ C(m)
$$

Memory maintenance rules:

$$
\begin{aligned}
\text{FWD} &: N_S(m)\ R(*) \rightarrow N_R(m)\ R(*)\\
\text{DELB} &: B(*) \rightarrow R(*)\\
\text{DELMS} &: M_s(*) \rightarrow R(*)
\end{aligned}
$$

Figure B.13: Additional rules for the Two-phase Intruder theory.

Since the intruder in our system has bounded memory, he should use it rationally. In particular, he should delete facts that are not useful for an attack, freeing some of his storage capacity for more useful information. This is formalized by using the memory management rules depicted in Figure B.12. Using these rules intruder can forget any facts stored in his memory which are of the form $M_?$. This contrast with [15], where these predicates were persistent throughout a run, that is, they were always present in the intruder's memory. Since in [15] intruder had unbounded memory, storing facts did not pose a problem.

In order to attack a protocol intruder does not need to digest every message put on the network. Furthermore, ignoring some messages can save intruder's memory. The FWD rule, for example, is a rule that is used to just forward sent messages to their destinations, and where the intruder does not learn any new data. That it, it just transforms a sent message $N_R(m)$ into a message $N_S(m)$ that can be received by other participants. Since this rule is not of the form of rules that belong to the memory maintenance theory, that is, its postcondition is not $R(*)$, for simplicity, we adapt Definition 6.6 to include this rule. Alternatively, in a trace we could simulate this rule with the following derivation:

$$N_S(m)\,R(*) \;\to_{REC}\; D(m)\,R(*) \;\to_{DM}\; M_s(m)\,R(*) \;\to_{USES}$$
$$M_s(m)\,C(m) \;\to_{SND}\; M_s(m)\,N_R(m) \;\to_{DELMS}\; N_R(m)\,R(*)$$

DM rule allows the intruder to remember complex sub-terms of a message being decomposed that might not be of interest at that moment, but that might be useful later. That can save memory when the intruder receives large submessages. It also is useful when intruder slightly modifies an intercepted messages, by using the USES rule, which allows the intruder to use complex terms in the composition phase. The DELD rule, on the other hand, allows the intruder to delete any decomposition fact, D, whenever it contains a message that is not useful to the intruder, such as data that he already knows. Therefore, with this rule, he does not need to expend his memory to further decompose such messages. It also reduces the number of steps, *i.e.*, the number of rules intruder has to perform to carry out an anomaly. Finally, the rule DELAB deletes auxiliary A facts and the rule DELMB deletes any M_c fact, freeing the intruder's memory.

Notice that in some rules we use the auxiliary predicate B, instead of the fact $R(*)$. This is a technicality in order to keep the intruder's theory two-phased, which will become clear after the following definitions. Intuitively, $B(*)$ facts represent "binned data" and can also be considered as empty facts. We therefore, from this point on, extend Definition 6.1 to consider the empty facts $B(*)$ as well and extend the weighting function by $\omega(B(*)) = 0$.

Remark. We restrict the type of facts the intruder is allowed to delete, *i.e.*we allow only the deletion of intruder's memory facts including auxiliary memory facts. Alternatively, we could also allow the intruder to delete public facts and in that way obstruct the normal protocol exchange. For example, deleting facts representing key distribution or participants' names or deleting role state predicates would exclude a principal form participating further in the protocol exchange. Even with above restrictions, we can still model such obstructions by the intruder, within his memory bounds, simply by removing messages (coming to and from a particular principal) from the network using REC rules.

Appendix C. Yahalom protocol

Yahalom is an authentication and secure key distribution protocol designed for use on an insecure network such as the internet. It involves a trusted server S. The protocol has been shown to be flawed by several authors.
The informal description of the protocol is given in figure C.14.

$$\begin{array}{l} A \longrightarrow B : A, n_a \\ B \longrightarrow S : B, \{A, n_a, n_b\}_{k_{BS}} \\ S \longrightarrow A : \{B, k_{AB}, n_a, n_b\}_{k_{AS}}, \{A, k_{AB}\}_{k_{BS}} \\ A \longrightarrow B : \{A, k_{AB}\}_{k_{BS}}, \{n_b\}_{k_{AB}} \end{array}$$

Figure C.14: Yahalom Protocol.

Symmetric keys k_{AS} and k_{BS} are shared between the server S and agents A and B, respectively. The server generates a fresh symmetric key k_{AB} which will be the session key to be shared between the two participants. Namely, the server sends to Alice a message containing the generated session key k_{AB} and a message to be forwarded to Bob.

A semi-founded protocol theory for the Yahalom protocol is given in Figure C.15.
Initial set of facts represents key distribution and announcement; 2 facts with keys for communication with the server and 2 facts for announcement of the participants' names:

$$W = Guy(A, k_{AS})\ Guy(B, k_{BS})\ AnnN(A)\ AnnN(B)\ .$$

There should be 3 additional facts for role states and another fact for the network predicate.

Therefore, a protocol run between A and B with no intruder involved requires a configuration of at least **8 facts of the size of at least 16**. The message that the server S sends to A has 15 symbols.

Role Regeneration Theory :
ROLA : $Guy(G, k_{GS})\ AnnN(G)\ P(*) \to Guy(G, k_{GS})\ AnnN(G)\ A_0(k_{GS})$
ROLB : $Guy(G, k_{GS})\ AnnN(G)\ P(*) \to Guy(g, k_{GS})\ AnnN(G)\ B_0(k_{GS})$
ROLS : $AnnN(G)\ P(*) \to AnnN(G)\ S_0()$
ERASEA : $A_2(k, G, x) \to P(*)$
ERASEB : $B_3(k, G, x, y) \to P(*)$
ERASES : $S_1(G, G') \to P(*)$

Protocol Theories $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{S}$:
A1 : $A_0(k_{GS})\ AnnN(G')\ P(*) \to \exists x.A_1(k_{GS}, G', x)\ N_S(\langle G, x\rangle)\ AnnN(G')$
A2 : $A_1(k_{GS}, G', x)\ N_R(\langle\ enc(k_{GS}, \langle G', \langle k_{GG'}, \langle x, y\rangle\rangle\rangle), z\ \rangle)$
$\quad \to A_2(k_{GS}, G', x, y)\ N_S(\langle\ z, enc(k_{GG'}, y)\ \rangle)$
B1 : $B_0(k_{GS})\ N_R(\langle G', x\rangle)\ AnnN(G')$
$\quad \to \exists y.B_1(k_{GS}, G', x, y)\ N_S(\langle\ G, enc(k_{GS}, \langle G', \langle x, y\rangle\rangle\ \rangle) AnnN(G')$
B2 : $B_1(k_{GS}, G', x, y)\ N_R(\langle\ enc(k_{GS}, \langle G', k_{G'G}\rangle), enc(k_{G'G}, y)\ \rangle))$
$\quad \to B_2(k_{GS}, G', x, y, k_{G'G})\ R(*)$
S1 : $S_0()\ Guy(G, k_{GS})\ Guy(G', k_{GS'})\ N_R(\langle\ G, enc(k_{GS}, \langle G', \langle x, y\rangle\rangle\ \rangle)$
$\quad \to \exists k_{G'G}.S_1(G', G)\ Guy(G, k_{GS})\ Guy(G', k_{GS'})$
$\qquad N_S(\langle\ enc(k_{G'S}, \langle G, \langle k_{G'G}, \langle x, y\rangle\rangle\rangle), enc(k_{GS}, \langle G', k_{G'G}\rangle)\ \rangle)$

Figure C.15: Semi-founded protocol theory for the Yahalom Protocol.

Appendix C.1. An attack on Yahalom Protocol

An anomaly on the Yahalom protocol is shown in Figure C.16. The attack assumes that the intruder knows the key k_{BS} shared between the server S and Bob. Intruder pretends to be Alice. He initiates the protocol by generating a nonce and sending it together with Alice's name to Bob. Since it is assumed that the intruder has the symmetric key k_{BS} that Bob shares with the server, intruder will be able do learn the nonce n_b. He can then compose a message that has the expected format of the last protocol message exchanged, *i.e.* the first part of the message is encrypted with the key k_{BS} and contains the freshly generated session key k_{AB}, and the second part of the message is the nonce n_b encrypted with that session key. Therefore intruder is able to trick Bob into thinking he had performed a valid protocol run with Alice and the trusted server. In reality Bob has only received messages from the intruder. The server hasn't been involved at all.

$$
\begin{array}{rcl}
I(A) \longrightarrow B: & & A, n_a \\
B \longrightarrow I(S): & & B, \{A, n_a, n_b\}_{k_{BS}} \\
\longrightarrow \quad : & & \text{omitted} \\
I(A) \longrightarrow B: & & \{A, n_a, n_b\}_{k_{BS}}, \{n_b\}_{n_a, n_b}
\end{array}
$$

Figure C.16: An attack on Yahalom Protocol.

Initial set of facts is: $W = Guy(A, k_{AS})\ Guy(B, k_{BS})\ AnnN(A)\ AnnN(B)$. For the symmetric encryption and decryption intruder uses rules ENCS and DECS. This attack requires encryption with a composed key so intruder needs ENCM rule for such encryption: ENCM : $C(x)C(y) \rightarrow M_k(x)C(enc(x, y))$.

The attack requires a configuration of at least **15** $R(*)$ **facts**; 6 for honest participants and 9 for the intruder. The protocol role predicates for Alice and Server are not used so 2 facts less are needed for honest participants.
The **size of the facts** should be at least **14** .
The trace with the anomaly is shown below.

$$
\begin{array}{l}
WB_0(k_{BS})M_g(A)M_k(k_{BS})R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*) \rightarrow_{USEG} \\
WB_0(k_{BS})M_g(A)M_k(k_{BS})C(A)R(*)R(*)R(*)R(*)R(*)R(*)P(*) \rightarrow_{GEN} \\
WB_0(k_{BS})M_g(A)M_k(k_{BS})C(A)M_n(n_a)R(*)R(*)R(*)R(*)R(*)P(*) \rightarrow_{USEN} \\
WB_0(k_{BS})M_g(A)M_k(k_{BS})C(A)M_n(n_a)C(n_a)R(*)R(*)R(*)R(*)P(*) \rightarrow_{COMP} \\
WB_0(k_{BS})M_g(A))M_k(k_{BS}) \\
\quad M_n(n_a)C(\langle A, n_a\rangle)R(*)R(*)R(*)R(*)R(*)P(*) \rightarrow_{SND} \\
WB_0(k_{BS})M_g(A)M_k(k_{BS})M_n(n_a) \\
\quad N_R(\langle A, n_a\rangle)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow_{DEL^2} \\
WB_0(k_{BS})M_k(k_{BS})N_R(\langle A, n_a\rangle)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow
\end{array}
$$

Bob receives the message intruder has sent and thinks it is a message from Alice, therefore sends a message to Server containing Alice's name.

$$
\begin{array}{l}
\rightarrow_{B1} WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})\ R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \\
\qquad N_S(\langle B, enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle)\rangle) \rightarrow
\end{array}
$$

Intruder intercepts the message intended for the server.

$$
\begin{array}{l}
\rightarrow_{REC} WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})\ R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*) \\
\quad D(\langle B, enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle)\rangle) \rightarrow_{DCMP} \\
\ WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})\ D(B) \\
\quad D(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))R(*)R(*)R(*)R(*)R(*)R(*)P(*) \rightarrow_{LRNG} \\
\ WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B) \\
\quad D(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))R(*)R(*)R(*)R(*)R(*)R(*)P(*) \rightarrow
\end{array}
$$

It is assumed that the intruder had previously learnt the key k_{BS} shared between the server and Bob, so he's able to decompose the encrypted submessage.

$$
\begin{array}{l}
\rightarrow_{DECS} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)\ P(*) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))D(\langle A, \langle n_a, n_b\rangle\rangle)R(*)R(*)R(*)R(*)R(*) \rightarrow_{DCMP} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)\ P(*) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))D(A)D(\langle n_a, n_b\rangle)R(*)R(*)R(*)R(*) \rightarrow_{DCMP} \\
WB_1(k_{BS}, A, n_a, n_b)M_k(k_{BS})M_g(B) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))D(A)D(n_a)D(n_b)R(*)R(*)R(*)P(*) \rightarrow_{LRNG} \\
WB_1(k_{BS}, A, n_a, n_b)M_k(k_{BS})M_g(B) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)D(n_a)D(n_b)R(*)R(*)R(*)P(*) \rightarrow_{LRNN} \\
WB_1(k_{BS}, A, n_a, n_b)M_k(k_{BS})M_g(B) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)D(n_b)R(*)R(*)R(*)P(*) \rightarrow_{LRNN} \\
WB_1(k_{BS}, A, n_a, n_b)M_k(k_{BS})M_g(B) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)M_n(n_b)R(*)R(*)R(*)P(*) \rightarrow
\end{array}
$$

Intruder starts composing the message that Bob expects to receive from Alice.

$$\begin{array}{l}
\to_{USEN} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)C(n_a) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)M_n(n_b)R(*)R(*)P(*) \to_{USEN} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS}))M_g(B)C(n_a)C(n_b) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)M_n(n_b)\ R(*)P(*) \to_{COMP} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)C(\langle n_a, n_b\rangle) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)M_n(n_b)R(*)R(*)P(*) \to_{USEG} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)C(\langle n_a, n_b\rangle)C(A) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)M_n(n_b)\ R(*)P(*) \to_{COMP} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)C(\langle A, \langle n_a, n_b\rangle\rangle) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)M_n(n_b)R(*)R(*)P(*) \to_{ENCS} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B) \\
\quad C(enc(k_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle)) \\
\quad M_g(A)M_n(n_a)M_n(n_b)\ R(*)R(*)P(*) \to_{USEN} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)M_g(A)M_n(n_a)M_n(n_b)C(n_a) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))C(enc(k_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))\ R(*)P(*) \to_{USEN} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)M_g(A)M_n(n_a)M_n(n_b) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))C(n_a)C(n_b)C(enc(k_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))\ P(*) \to
\end{array}$$

Notice there are no $R(*)$ facts in the configuration.

$$\begin{array}{l}
\to_{COMP} \\
WB_1(k_{BS}, A, n_a, n_b)\ M_k(k_{BS})M_g(B)\ C(enc(k_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))R(*)P(*) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))\ M_g(A)M_n(n_a)M_n(n_b)C(\langle n_a, n_b\rangle) \to_{USEN} \\
WB_1(k_{BS}, A, n_a, n_b)M_k(k_{BS})M_g(B)C(n_b)C(enc(k_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))P(*) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)M_n(n_b)C(\langle n_a, n_b\rangle) \to
\end{array}$$

He uses the composed key for encryption to compose the message that matches the format that Bob expects to receive.

$$\begin{array}{l}
\to_{ENCM} \\
WB_1(k_{BS}, A, n_a, n_b)M_k(k_{BS})M_g(B)M_k(\langle n_a, n_b\rangle) \\
\quad M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))M_g(A)M_n(n_a)M_n(n_b) \\
\quad C(enc(\langle n_a, n_b\rangle, n_b))C(enc(k_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))P(*) \to_{COMP} \\
WB_1(k_{BS}, A, n_a, n_b)M_k(k_{BS})M_g(B)M_g(A)M_k(\langle n_a, n_b\rangle) \\
\quad M_n(n_a)M_n(n_b)M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle)) \\
\quad C(\langle enc(k_{BS}, \langle A, \langle n_a, n_b\rangle\rangle), enc(\langle n_a, n_b\rangle, n_b)\rangle)R(*)P(*) \to
\end{array}$$

$$
\begin{aligned}
&\rightarrow_{SND}\\
&WB_1(k_{BS}, A, n_a, n_b)M_k(k_{BS})M_g(B)M_g(A)M_k(\langle n_a, n_b\rangle)\\
&\quad M_n(n_a)M_n(n_b)M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))\\
&\quad N_R(\langle enc(k_{BS}, \langle A, \langle n_a, n_b\rangle\rangle), enc(\langle n_a, n_b\rangle, n_b)\rangle)\ R(*)R(*) \rightarrow
\end{aligned}
$$

Bob receives what he believes is a message from Alice containing the session key freshly generated by the server. Therefore he stores the false key and thinks he had completed a successful protocol run with Alice.

$$
\begin{aligned}
&\rightarrow_{B2}\\
&WB_2(k_{BS}, A, n_a, n_b, \langle n_a, n_b\rangle)M_k(k_{BS})M_g(B)M_k(\langle n_a, n_b\rangle)\\
&\quad M_g(A)M_n(n_a)M_n(n_b)M_c(enc(K_{BS}, \langle A, \langle n_a, n_b\rangle\rangle))R(*)R(*)P(*)
\end{aligned}
$$

Appendix D. Otway-Rees Protocol

The Otway-Rees Protocol is another well-known protocol that has been shown to be flawed. It's informal description is depicted in Figure D.17.

$$
\begin{array}{l}
A \longrightarrow B : M, A, B, \{n_a, M, A, B\}_{k_{AS}} \\
B \longrightarrow S : M, A, B, \{n_a, M, A, B\}_{k_{AS}}, \{n_b, M, A, B\}_{k_{BS}} \\
S \longrightarrow B : M, \{n_a, k_{AB}\}_{k_{AS}}, \{n_b, k_{AB}\}_{k_{BS}} \\
B \longrightarrow A : M, \{n_a, k_{AB}\}_{k_{AS}}
\end{array}
$$

Figure D.17: Otway-Rees Protocol.

The protocol also involves a trusted server. Keys k_{AS} and k_{BS} are symmetric keys for communication of the participants with the server. In the above protocol specification M is a nonce (a run identifier). A semi-founded protocol theory for Otway-Rees protocol is given in Figure D.18.

Initiator A sends to B the nonce M and names A and B unencrypted together with an encrypted message readable only by the server S of the form shown. B forwards the message to S together with a similar encrypted component. The server S decrypts the message components and checks that the components match. If so, then it generates a key $k_{A,B}$ and sends message to B, who then forwards part of this message to A. A and B will use the key $k_{A,B}$ only if the message components generated by the server S contain the correct nonces n_a and n_b respectively.

Initial set of facts represents key distribution and announcement; 2 facts with keys for communication with the server and 2 facts for announcement of the participants' names: $W = Guy(A, K_{AS})\ Guy(B, k_{BS})\ AnnN(A)\ AnnN(B)$. There should be additional 3 facts for role states and another fact for the network predicate. Therefore, a protocol run between A and B with no intruder involved requires a configuration of at least **8 facts of the size of at least 26**. The fact representing the network message that the B sends to S has 25 symbols.

Role Regeneration Theory :
ROLA : $Guy(G, k_{GS})\ AnnN(G)\ P(*) \to Guy(G, k_{GS})\ AnnN(G)\ A_0(k_{GS})$
ROLB : $Guy(G, k_{GS})\ AnnN(G)\ P(*) \to Guy(G, k_{GS})\ AnnN(G)\ B_0(k_{GS})$
ROLS : $P(*) \to\ S_0()$
ERASEA : $A_2(k, G, x, y, k') \to P(*)$
ERASEB : $B_2(k, G, x, y, z, w, k') \to P(*)$
ERASES : $S_1(G, G') \to P(*)$

Protocol Theories $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{S}$:
A1 : $A_0(k_{GS})\ AnnN(G')\ P(*) \to \exists x.y.A_1(k_{GS}, G', x, y)\ AnnN(G')$
$\quad N_S(\langle x, \langle G, \langle G', enc(k_{GS}, \langle y, \langle x, \langle G, G'\rangle\rangle\rangle)\rangle\rangle\rangle)$
A2 : $A_1(k_{GS}, G', x, y)\ N_R(\langle x,\ enc(k_{GS}, \langle y, \langle k_{GG'}\rangle)\ \rangle)$
$\quad \to A_2(k_{GS}, G', x, y, k_{GG'})\ P(*)$
B1 : $B_0(k_{GS})\ AnnN(G')\ N_R(\langle x, \langle G', \langle G, z\rangle\rangle\rangle)$
$\quad \to \exists w.B_1(k_{GS}, G', x, z, w)\ AnnN(G')$
$\quad N_S(\langle x, \langle G', \langle G, \langle z, enc(k_{GS}, \langle w, \langle x, \langle G', G\rangle\rangle\rangle)\rangle\rangle\rangle\rangle)$
B2 : $B_1(k_{GS}, G', x, z, w)\ N_R(\langle\ x, \langle t, enc(k_{GS}, \langle w, k_{GG'}\rangle)\rangle\ \rangle)$
$\quad \to B_2(k_{GS}, G', x, z, w, t, k_{GG'})\ N(\langle x, t\rangle)$
S1 : $S_0()\ Guy(G, k_{GS})\ Guy(G', k_{GS'})$
$\quad N_R(\langle x, \langle G, \langle G', \langle\ enc(k_{GS}, \langle y, \langle x, \langle G, G'\rangle\rangle\rangle), enc(k_{G'S}, \langle w, \langle x, \langle G, G'\rangle\rangle\rangle)\ \rangle\ \rangle\rangle\rangle)$
$\quad \to \exists k_{GG'}.S_1(G, G')\ Guy(G, k_{GS})\ Guy(G', k_{GS'})$
$\quad N_S(\langle\ x, \langle enc(k_{GS}, \langle y, k_{GG'}\rangle), enc(k_{G'S}, \langle w, k_{GG'}\rangle)\rangle\ \rangle)$

Figure D.18: Semi-founded protocol theory for the Otway-Rees Protocol.

Appendix D.1. A type flaw attack on Otway-Reese Protocol

In this anomaly, shown in Figure D.19, principal A is fooled into believing that the triple $\langle M, A, B\rangle$ is in fact the new key. This triple is of course public knowledge. This is an example of a type flaw. It is also possible to wait until B sends the second message of the original protocol and then reflect appropriate components back to both A and B and then monitor the conversation between them.

$$\begin{array}{ll} A \longrightarrow I(B): & M, A, B, \{n_a, M, A, B\}_{k_{AS}} \\ I(B) \longrightarrow A: & M, \{n_a, M, A, B\}_{k_{AS}} \end{array}$$

Figure D.19: A type-flaw attack on Otway-Rees Protocol.

Intruder intercepts Alice's message and replies with a message of the format Alice expects to receive from Bob containing the fresh key. She gets the "key" $\langle M, \langle A, B\rangle\rangle$ that is the public knowledge, not a secret. Neither Bob nor the server get involved.

Initial set of facts is:

$$W = Guy(A, K_{AS})\ Guy(B, k_{BS})\ AnnN(A)\ AnnN(B)\ .$$

The trace representing the anomaly is shown below.

$$\begin{array}{l}
WA_0(k_{AS})\ R(*)R(*)R(*)R(*)P(*) \to_{A1} \\
WA_1(k_{AS}, B, M, n_a)\ R(*)R(*)R(*)R(*)R(*) \\
\quad N_S(\langle M, \langle A, \langle B, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle\rangle\rangle) \to_{REC} \\
WA_1(k_{AS}, B, M, n_a)\ R(*)R(*)R(*)R(*)P(*) \\
\quad D(\langle M, \langle A, \langle B, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle\rangle\rangle) \to_{DCMP} \\
WA_1(k_{AS}, B, M, n_a)\ R(*)R(*)R(*)P(*) \\
\quad D(M)\ D(\langle A, \langle B, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle\rangle) \to_{DCMP} \\
WA_1(k_{AS}, B, M, n_a)\ R(*)R(*)P(*) \\
\quad D(M)\ D(A)\ D(\langle B, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle) \to_{DELD} \\
WA_1(k_{AS}, B, M, n_a)\ R(*)R(*)P(*) \\
\quad D(M)\ B(*)\ D(\langle B, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle) \to_{DCMPB} \\
WA_1(k_{AS}, B, M, n_a)\ R(*)R(*)P(*) \\
\quad D(M)\ D(B)\ D(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \to
\end{array}$$

$$
\begin{array}{l}
\rightarrow_{DELD} \\
WA_1(k_{AS}, B, M, n_a)\ R(*)R(*)P(*) \\
\quad D(M)\ B(*)\ D(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \rightarrow_{DM} \\
WA_1(k_{AS}, B, M, n_a)\ R(*)R(*)P(*) \\
\quad D(M)\ B(*)\ M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \rightarrow_{LRNN} \\
WA_1(k_{AS}, B, M, n_a)\ M_n(M)\ B(*)R(*)R(*)P(*) \\
\quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \rightarrow_{USEN} \\
WA_1(k_{AS}, B, M, n_a)\ M_n(M)\ C(M)\ B(*)R(*)P(*) \\
\quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \rightarrow_{USEC} \\
WA_1(k_{AS}, B, M, n_a)\ M_n(M)\ C(M)\ \ B(*)P(*) \\
\quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ C(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \rightarrow
\end{array}
$$

Notice that there are no $R(*)$ facts in the configuration.

$$
\begin{array}{l}
\rightarrow_{COMP} \\
WA_1(k_{AS}, B, M, n_a)\ M_n(M)\ M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \\
\qquad C(\langle M, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle)\ B(*)R(*)P(*) \rightarrow_{SND} \\
WA_1(k_{AS}, B, M, n_a)\ M_n(M)\ M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \\
\quad N_R(\langle M, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle)\ B(*)R(*)R(*) \rightarrow_{A2} \\
WA_2(k_{AS}, B, M, n_a, \langle M, \langle A, B\rangle\rangle\ )\ M_n(M)\ B(*)R(*)R(*)P(*) \\
\quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))
\end{array}
$$

This attack requires a configuration of at least **<u>11 facts</u>** in total: $6P(*)$ facts (for the honest participants) and 5 $R(*)$ facts (for the intruder).
The **<u>size of facts has to be at least 15</u>**.
Although some protocol messages were not sent it could be reasonable to allow a normal protocol execution. *i.e.*to require the facts to have size of at least 25 slots for constant names. However, in the attack itself, the messages sent have the size of at most 14 symbols. Additional 1 counts for the predicate name.

This type of anomalies can be prevented by a typed alphabet. Since we allow only atomic keys within our typed alphabet this attack is not possible. The tuple of terms $\langle M, A, B\rangle$ cannot be confused with a term of type ”key”.

Appendix D.2. Replay attack on Otway-Reese Protocol

This attack was presented by Wang and Qing (Two new attacks on Otway-Rees Protocol, In: IFIP/SEC2000, Beijing: International Academic Publishers, 2000. 137-139.).

It is a replay anomaly, that is, an intruder overhears a message in a protocol session and can therefore replay this message or some of its parts to form messages of the expected protocol form, later, in another protocol session and trick an honest participant.

$$
\begin{aligned}
A \longrightarrow B &: M, A, B, \{n_a, M, A, B\}_{k_{AS}} \\
B \longrightarrow S &: M, A, B, \{n_a, M, A, B\}_{k_{AS}}, \{n_b, M, A, B\}_{k_{BS}} \\
S \longrightarrow (B)I &: M, \{n_a, k_{AB}\}_{k_{AS}}, \{n_b, k_{AB}\}_{k_{BS}} \\
I(B) \longrightarrow S &: M, A, B, \{n_a, M, A, B\}_{k_{AS}}, \{n_b, M, A, B\}_{k_{BS}} \\
S \longrightarrow B(I) &: M, \{n_a, k'_{AB}\}_{k_{AS}}, \{n_b, k'_{AB}\}_{k_{BS}} \\
I(S) \longrightarrow B &: M, \{n_a, k_{AB}\}_{k_{AS}}, \{n_b, k'_{AB}\}_{k_{BS}} \\
B \longrightarrow A &: M, \{n_a, k_{AB}\}_{k_{AS}}
\end{aligned}
$$

Figure D.20: Replay attack on Otway-Rees Protocol.

As shown in figure D.20, intruder intercepts a request to the server and stores data so he's able to replay the message. The server responds to a replayed request generating a fresh session key. Intruder is able to modify the messages so that Alice and Bob get different keys.

Alice and Bob start the protocol. Intruder copies the message that Bob sends to the server and then he replays it later. The attack is successful if the server cannot recognize duplicate requests.

When the attack run is over, Alice and Bob do get the session keys, but they get two different ones; Alice gets k_{AB} and Bob gets k'_{AB}.

This attack requires a configuration of at least **17 facts** in total: $8P(*)$ facts (for the honest participants) and 9 $R(*)$ facts (for the intruder).
The **size of facts has to be at least 26**.

Initial set of facts is: $W = Guy(A, K_{AS})\, Guy(B, k_{BS})\, AnnN(A)\, AnnN(B)$.

The trace representing the anomaly is shown below.

Alice starts a protocol session by sending the first protocol message to Bob.

$$\begin{array}{l} WA_0(k_{AS})\ B_0(k_{BS})\ S_0()\ R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*) \to_{A1} \\ WA_1(k_{AS}, B, M, n_a)\ B_0(k_{BS})\ S_0()\ R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \\ \quad N_S((\langle M, \langle A, \langle B, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle\rangle\rangle) \end{array}$$

Intruder does not need data from this message, so he simply forwards it to Bob.

$$\begin{array}{l} \to_{FWD} \\ WA_1(k_{AS}, B, M, n_a)\ B_0(k_{BS})\ S_0()\ R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \\ \quad N_R(\langle M, \langle A, \langle B, enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle\rangle\rangle) \to_{B1} \\ WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b)\ S_0() \\ \quad N_S(\langle M, \langle A, \langle B, \langle enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle), enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)\rangle\rangle\rangle\rangle) \\ \quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \to \end{array}$$

Bob responds. This time intruder needs to intercept the message to store the message parts in order to replay this message to the server later on. Intruder performs a normalized derivation and deletes unnecessary data.
For simplicity, we use $z = (enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)$.

$$\begin{array}{l} \to_{REC} \\ WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b)\ S_0() \\ \quad D(\langle M, \langle A, \langle B, \langle enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle), enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)\rangle\rangle\rangle\rangle) \\ \quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*) \to_{DCMP^4} \\ WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0() \\ \quad D(M)\ D(A)\ D(B)\ R(*)R(*)R(*)R(*)P(*) \\ \quad D(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\ D(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle) \\ \to_{(LRNN, LRNG, LRNG, DM^2)} \\ WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_n(M)\ M_g(A)\ M_g(B) \\ \quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ R(*)R(*)R(*)R(*) \\ \quad M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)\ P(*) \to_{USES^2} \\ WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_n(M)\ M_g(A)\ M_g(B) \\ \quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\ M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)\ P(*) \\ \quad C(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)\ C(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\ R(*)R(*) \to_{COMP} \\ WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_n(M)\ M_g(A)\ M_g(B)\ P(*) \\ \quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\ M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)\ R(*)R(*) \\ \quad C(\langle enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle), enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle)R(*) \to_{USEG} \\ WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_n(M)\ M_g(A)\ M_g(B) \\ \quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\ C(M) \\ \quad M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)\ R(*)R(*)P(*) \\ \quad C(\langle enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle), enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)\rangle) \to \end{array}$$

$$
\begin{aligned}
&\rightarrow_{COMP}\\
&WA_1(k_{AS},B,M,n_a)B_1(k_{BS},A,M,z,n_b)\ S_0()\ M_n(M)\ M_g(A)\ M_g(B)\\
&\quad M_s(enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle))\\
&\quad M_s(enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle))\ R(*)R(*)R(*)P(*)\\
&\quad C(\langle M,\langle enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle),enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle)\rangle\rangle)\ \rightarrow_{SND}\\
&WA_1(k_{AS},B,M,n_a)B_1(k_{BS},A,M,z,n_b)\ S_0()\ M_n(M)\ M_g(A)\ M_g(B)\\
&\quad M_s(enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle))\\
&\quad M_s(enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle))\ R(*)R(*)R(*)R(*)\\
&\quad N_R(\langle M,\langle enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle),enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle)\rangle\rangle)\ \rightarrow
\end{aligned}
$$

Intruder has to be careful with deletion rules, since he will need some knowledge for reproducing messages later in the protocol attack.

$$
\begin{aligned}
&\rightarrow_{DEL^3}\\
&WA_1(k_{AS},B,M,n_a)\ B_1(k_{BS},A,M,z,n_b)\ S_0()\ M_g(A)M_g(B)\\
&\quad M_s(enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle))\ M_s(enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle))\\
&\quad R(*)R(*)R(*)R(*)R(*)\\
&\quad N_R(\langle M,\langle A,\langle B,\langle enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle),enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle)\rangle\rangle\rangle\rangle)\ \rightarrow
\end{aligned}
$$

The server responds to the request and finishes the session by deleting its final role state predicate and creating an initial role state for the new session.

$$
\begin{aligned}
&\rightarrow_{S1}\ WA_1(k_{AS},B,M,n_a)\ B_1(k_{BS},A,M,z,n_b)\ S_1(A,B)\ M_g(A)M_g(B)\\
&\quad M_s(enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle))\ M_s(enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle))\\
&\quad R(*)R(*)R(*)R(*)R(*)\\
&\quad N_S(\langle\ M,\langle enc(k_{AS},\langle n_a,k_{AB}\rangle),enc(k_{BS},\langle n_b,k_{AB}\rangle)\rangle\ \rangle)\\
&\rightarrow_{ERASES,ROLS}\\
&WA_1(k_{AS},B,M,n_a)\ B_1(k_{BS},A,M,z,n_b)\ S_0()\ M_g(A)M_g(B)\\
&\quad M_s(enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle))\ M_s(enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle))\\
&\quad R(*)R(*)R(*)R(*)R(*)\\
&\quad N_S(\langle\ M,\langle enc(k_{AS},\langle n_a,k_{AB}\rangle),enc(k_{BS},\langle n_b,k_{AB}\rangle)\rangle\ \rangle)\ \rightarrow
\end{aligned}
$$

Intruder removes the message server has sent so Bob never receives it. He replays Bob's request message using the data he had learnt from Bob's original request.

$$
\begin{aligned}
&\rightarrow_{REC}\\
&WA_1(k_{AS},B,M,n_a)\ B_1(k_{BS},A,M,z,n_b)\ S_0()\ M_g(A)M_g(B)R(*)R(*)\\
&\quad M_s(enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle))\ M_s(enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle))\ R(*)R(*)\\
&\quad D(\langle\ M,\langle enc(k_{AS},\langle n_a,k_{AB}\rangle),enc(k_{BS},\langle n_b,k_{AB}\rangle)\rangle\ \rangle)\ P(*)\ \rightarrow_{DCMP}\\
&WA_1(k_{AS},B,M,n_a)\ B_1(k_{BS},A,M,z,n_b)\ S_0()\ M_g(A)M_g(B)\ R(*)R(*)R(*)\\
&\quad M_s(enc(k_{AS},\langle n_a,\langle M,\langle A,B\rangle\rangle\rangle))\ M_s(enc(k_{BS},\langle n_b,\langle M,\langle A,B\rangle\rangle\rangle))\\
&\quad D(M)D(\langle enc(k_{AS},\langle n_a,k_{AB}\rangle),enc(k_{BS},\langle n_b,k_{AB}\rangle)\rangle)\ P(*)\ \rightarrow
\end{aligned}
$$

$$
\begin{array}{l}
\to_{LRNN} \\
WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_g(A)M_g(B)\ R(*)R(*)R(*) \\
\quad M_n(M)\ M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)) \\
\quad D(\langle enc(k_{AS}, \langle n_a, k_{AB}\rangle), enc(k_{BS}, \langle n_b, k_{AB}\rangle)\rangle)\ P(*) \to_{DCMP} \\
WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_g(A)M_g(B)\ R(*)R(*) \\
\quad M_n(M)\ M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)) \\
\quad D(enc(k_{AS}, \langle n_a, k_{AB}\rangle))\ D(enc(k_{BS}, \langle n_b, k_{AB}\rangle))\ P(*) \to_{DELD} \\
WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_g(A)M_g(B)\ R(*)R(*) \\
\quad M_n(M)\ M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)) \\
\quad D(enc(k_{AS}, \langle n_a, k_{AB}\rangle))\ B(*)P(*) \to_{DM} \\
WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_g(A)M_g(B)\ R(*)R(*) \\
\quad M_n(M)\ M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)) \\
\quad M_s(enc(k_{AS}, \langle n_a, k_{AB}\rangle))\ B(*)P(*) \to_{(USES^2)} \\
WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_g(A)M_g(B) \\
\quad M_n(M)\ M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ M_s(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)) \\
\quad M_s(enc(k_{AS}, \langle n_a, k_{AB}\rangle))\ B(*)P(*) \\
\quad C(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ C(enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)) \to
\end{array}
$$

Notice that at this point there are no $R(*)$ facts in the configuration. Intruder continues to compose the request message, sends it to the server and deletes unnecessary data from his memory.

$$
\begin{array}{l}
\to_{(COMP,USEG,COMP,USEG,COMP,USEN,COMP,SND,DEL^5)} \\
WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b))\ S_0()\ M_c(enc(k_{AS}, \langle n_a, k_{AB}\rangle)) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad N_R(\langle M, \langle A, \langle B, \langle enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle, enc(k_{BS}, \langle n_b, \langle M, \langle A, B\rangle\rangle\rangle)\rangle\rangle\rangle\rangle) \to
\end{array}
$$

The Server does not detect the replay message and replies with a fresh message containing a new key k'_{Ab}. Intruder intercepts second server's reply and sends a modified message to Bob. That is an incorrect protocol message but Bob cannot detect it.

$$
\begin{array}{l}
\to_{(S1,ERASES)} \\
WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_s(enc(k_{AS}, \langle n_a, k_{AB}\rangle)) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad N_S(\langle\ M, \langle enc(k_{AS}, \langle n_a, k'_{AB}\rangle), enc(k_{BS}, \langle n_b, k'_{AB}\rangle)\rangle\ \rangle) \to
\end{array}
$$

Intruder intercepts the second reply from the Server, switches submessages and sends the modified message to Bob.

$$
\begin{aligned}
&\rightarrow_{REC} \\
&WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_s(enc(k_{AS}, \langle n_a, k_{AB}\rangle)) \\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*) \\
&\quad D(\langle\ M, \langle enc(k_{AS}, \langle n_a, k'_{AB}\rangle), enc(k_{BS}, \langle n_b, k'_{AB}\rangle)\rangle\ \rangle) \rightarrow_{DCMP^2} \\
&WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_c(enc(k_{AS}, \langle n_a, k_{AB}\rangle)) \\
&\quad D(M)\ D(enc(k_{AS}, \langle n_a, k'_{AB}\rangle))\ D(enc(k_{BS}, \langle n_b, k'_{AB}\rangle)) \\
&\quad R(*)R(*)R(*)R(*)R(*)P(*) \rightarrow_{(LRNN, DELD, LRNA)} \\
&WA_1(k_{AS}, B, M, n_a)B_1(k_{BS}, A, M, z, n_b)\ S_0()\ M_c(enc(k_{AS}, \langle n_a, k_{AB}\rangle)) \\
&\quad M_n(M)\ B(*)\ M_c(enc(k_{BS}, \langle n_b, k'_{AB}\rangle))\ A(enc(k_{BS}, \langle n_b, k'_{AB}\rangle)) \\
&\quad R(*)R(*)R(*)R(*)P(*) \rightarrow_{(USEC^2, COMP, USEN, COMP, SND, DEL^5)} \\
&WA_1(k_{AS}, B, M, n_a)\ B_1(k_{BS}, A, M, z, n_b)\ S_0() \\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \\
&\quad N_R(\langle\ M, \langle enc(k_{AS}, \langle n_a, k_{AB}\rangle), enc(k_{BS}, \langle n_b, k'_{AB}\rangle)\rangle\ \rangle) \rightarrow
\end{aligned}
$$

Bob receives a message that looks like the normal server's reply and sends the next message to Alice. For simplicity, we use $t = enc(k_{AS}, \langle n_a, k_{AB}\rangle)$.

$$
\begin{aligned}
&\rightarrow_{B2} \\
&WA_1(k_{AS}, B, M, n_a)B_2(k_{BS}, A, M, z, n_b, t, k'_{AB})\ S_0() \\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \\
&\quad N_S(\langle\ M, enc(k_{AS}, \langle n_a, k_{AB}\rangle)\ \rangle) \rightarrow
\end{aligned}
$$

Intruder simply forwards the message to Alice, who receives it and moves into final state believing she and Bob now share a fresh session key.

$$
\begin{aligned}
&\rightarrow_{FWD} \\
&WA_1(k_{AS}, B, M, n_a)\ B_2(k_{BS}, A, M, z, n_b, t, k'_{AB})\ S_0() \\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \\
&\quad N_R(\langle\ M, enc(k_{AS}, \langle n_a, k_{AB}\rangle)\ \rangle) \rightarrow_{A2} \\
&WA_2(k_{AS}, B, M, n_a, k_{AB})\ B_2(k_{BS}, A, M, z, n_b, t, k'_{AB}))\ S_0() \\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*)
\end{aligned}
$$

As a result both Alice and Bob do get the session key, but they get different keys; Alice get k_{AB} while Bob gets k'_{AB}.

Appendix E. Woo-Lam protocol, simplified

The informal description of this one-way authentication protocol is shown in Figure E.21.

$$
\begin{array}{l}
A \longrightarrow B : A \\
B \longrightarrow A : n_b \\
A \longrightarrow B : \{n_b\}_{k_{AS}} \\
B \longrightarrow S : \{A, \{n_b\}_{k_{AS}}\}_{k_{BS}} \\
S \longrightarrow B : \{A, n_b\}_{k_{BS}}
\end{array}
$$

Figure E.21: Simplified Woo-Lam Protocol.

Woo and Lam presented this authentication protocol using symmetric cryptography in which Alice tries to prove her identity to Bob using a trusted third party, the server S. Firstly, Alice claims her identity. In response, Bob generates a nonce. Alice then returns this challenge encrypted with the secret symmetric key k_{AS} that she shares with the server. Bob passes this to server for translation and then the server returns the nonce received to Bob. Both bob and the server use the shared symmetric key k_{BS} for that communication. Finally, Bob verifies the nonce.

The Woo-Lam protocol in its various versions appear to be subject to various attacks.

A semi-founded protocol theory for the Woo-Lam protocol is given in Figure E.22.

Initial set of facts represents key distribution and announcement. It includes 2 facts with keys for communication with the server and 2 facts for announcement of the participants' names:

$$W = Guy(A, K_{AS})\ Guy(B, k_{BS})\ AnnN(A)\ AnnN(B)\ .$$

There should be additional 2 facts for role states and another fact for the network predicate. Therefore, a protocol run between A and B with no intruder involved requires a configuration of at least **7 facts of the size of at least 6**.

Role Regeneration Theory :

$$
\begin{array}{rl}
\text{ROLA}: & Guy(G,k_{GS})\ AnnN(G)P(*) \rightarrow Guy(G,k_{GS})\ AnnN(G)A_0(k_{GS}) \\
\text{ROLB}: & Guy(G,k_{GS})\ AnnN(G)P(*) \rightarrow Guy(G,k_{GS})\ AnnN(G)B_0(k_{GS}) \\
\text{ERASEA}: & A_2(k,G,x) \rightarrow P(*) \\
\text{ERASEB}: & B_3(k,G,x,y) \rightarrow P(*)
\end{array}
$$

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

$$
\begin{array}{rl}
\text{A1}: & A_0(k_{GS})\ AnnN(G')P(*) \rightarrow A_1(k_{GS},G')\ N_S(G)\ AnnN(G') \\
\text{A2}: & A_1(k_{GS},G')\ N_R(x) \rightarrow A_2(k_{GS},G',x)N_S(enc(k_{GS},x)) \\
\text{B1}: & B_0(k_{GS})\ N_R(G')\ AnnN(G') \\
 & \quad \rightarrow \exists x.B_1(k_{GS},G',x)\ N_S(x)\ AnnN(G') \\
\text{B2}: & B_1(k_{GS},G',x)\ N_R(y) \rightarrow B_2(k_{GS},G',x,y)\ N_S(enc(k_{GS},\langle G',y\rangle)) \\
\text{B3}: & B_2(k_{GS},G',x,y)\ N_R(enc(k_{GS},x)) \rightarrow B_3(k_{GS},G',x,y)\ P(*) \\
\text{S1}: & N_R(enc(k_{GS},\langle G',enc(K_{GS'},x)\rangle))\ Guy(G,k_{GS})\ Guy(G',k_{GS'}) \\
 & \quad \rightarrow N_S(enc(k_{GS},x))\ Guy(G,k_{GS})\ Guy(G',k_{GS'})
\end{array}
$$

Figure E.22: Semi-founded protocol theory for the simplified Woo-Lam Protocol.

Appendix E.1. An attack on simplified Woo-Lam protocol

An anomaly on Woo-Lam protocol in shown in Figure E.23.

$$\begin{array}{ll} I(A) \longrightarrow B: & A \\ B \longrightarrow I(A): & n_b \\ I(A) \longrightarrow B: & n_b \\ B \longrightarrow I(S): & \{A, n_b\}_{k_{BS}} \\ I(S) \longrightarrow B: & \{A, n_b\}_{k_{BS}} \end{array}$$

Figure E.23: An attack on simplified Woo-Lam Protocol.

Intruder pretends to be Alice and sends Alice's name to Bob. Bob replies and than receives a message that he believes comes from Alice therefore he encrypts it with his key. Than the intruder send the message that looks like the valid server's reply. Bob finishes the role thinking he had completed a successful protocol run with Alice. Neither Alice nor the server were involved. Intruder initiates the protocol impersonating Alice. Then he also impersonates the server and although intruder does not know the keys shared between the server and Alice and Bob, respectively, he is able to trick Bob into thinking that he had completed a proper protocol exchange with Alice.

Initial set of facts is $W = Guy(A, K_{AS})\ Guy(B, k_{BS})\ AnnN(A)\ AnnN(B)$. This attack requires a configuration of at least **<u>11 facts</u>** (6 for the protocol and additional 2 for the intruder) of the **<u>size 6</u>** . Notice that we did not need the role state predicate for Alice, therefore the protocol did not require the usual 7 facts.

$$\begin{array}{l} W\ B_0(k_{BS})\ M_g(A)\ R(*)\ P(*) \rightarrow_{USEG} \\ W\ B_0(k_{BS})\ M_g(A)\ C(A)\ P(*) \rightarrow_{SND} \\ W\ B_0(k_{BS})\ M_g(A)\ N_R(A)\ R(*) \rightarrow_{B1} \\ W\ B_1(k_{BS}, A, n_b)\ M_g(A)\ N_S(n_b)\ R(*) \rightarrow_{FWD} \\ W\ B_1(k_{BS}, A, n_b)\ M_g(A)\ N_R(n_b)\ R(*) \rightarrow_{B2} \\ W\ B_2(k_{BS}, A, n_b, n_b)\ M_g(A)\ N_S(enc(k_{BS}, \langle A, n_b \rangle))\ R(*) \rightarrow_{FWD} \\ W\ B_2(k_{BS}, A, n_b, n_b)\ M_g(A))\ N_R(enc(k_{BS}, \langle A, n_b \rangle))\ R(*) \rightarrow_{B3} \\ W\ B_3(k_{BS}, A, n_b, n_b)\ M_g(A)\ R(*) P(*) \end{array}$$

This attack requires a configuration of at least **<u>8 facts</u>** (6 for the protocol and additional 2 for the intruder) of the **<u>size 6</u>** .

Appendix F. An audited key distribution protocol from MSR

The following protocol was introduced in [15]. It is a fragment of an audited key distribution protocol, for one key server and s clients. The protocol assumes that a private symmetric key K is shared between the principals $A, B_1, \ldots ; B_s$ and C. Here A is a key server, $B_1; \ldots , B_s$ are clients, and C is an audit process. There are s Server/Client sub-protocols, one for each client. In these sub-protocols A sends a value which corresponds to a certain binary pattern, and B_i responds by incrementing the pattern by one. We use the notation x_i to indicate the "don't care" values in the messages in the Server/Client sub-protocols.

We show the protocol for $s = 4$.

Keys: K - symmetric encryption key shared by A, B_i, C

Server / Client Protocols

$A \longrightarrow B_1 \ : \{x_1, x_2, x_3, 0\}_K$
$B_1 \longrightarrow A \ : \{x_1, x_2, x_3, 1\}_K$

$A \longrightarrow B_2 \ : \{x_1, x_2, 0, 1\}_K$
$B_2 \longrightarrow A \ : \{x_1, x_2, 1, 0\}_K$

$A \longrightarrow B_3 \ : \{x_1, 0, 1, 1\}_K$
$B_3 \longrightarrow A \ : \{x_1, 1, 0, 0\}_K$

$A \longrightarrow B_4 \ : \{0, 1, 1, 1\}_K$
$B_4 \longrightarrow A \ : \{1, 0, 0, 0\}_K$

Audit Protocols

$A \longrightarrow C \ : \{0, 0, 0, 0\}_K$
$C \longrightarrow A \ :$ OK

$A \longrightarrow C \ : \{1, 1, 1, 1\}_K$
$C \longrightarrow A \ :$ SECRET

Figure F.24: Exponential Protocol

The protocol also includes two audit sub-protocols. In the first audit protocol the server A sends a message of all zero's to C to indicate that the protocol finished correctly. In the second audit protocol, A sends a message of all one's to indicate that there is an error. The second audit protocol has the side-effect of broadcasting the SECRET if C receives the error message.

$$
\begin{array}{llll}
\multicolumn{4}{l}{\text{Role regeneration theory :}} \\
\text{ROLA :} & P(*) \rightarrow A_0(K) & \text{ERASEA :} & A_4(K) \rightarrow P(*) \\
\text{ROLB1 :} & P(*) \rightarrow B1_0(K) & \text{ERASEB1 :} & B1_1(K) \rightarrow P(*) \\
\text{ROLB2 :} & P(*) \rightarrow B2_0(K) & \text{ERASEB2 :} & B2_1(K) \rightarrow P(*) \\
\text{ROLB3 :} & P(*) \rightarrow B3_0(K) & \text{ERASEB3 :} & B3_1(K) \rightarrow P(*) \\
\text{ROLB4 :} & P(*) \rightarrow B4_0(K) & \text{ERASEB4 :} & B4_1(K) \rightarrow P(*) \\
\text{ROLC :} & P(*) \rightarrow C_0(K) & \text{ERASEC :} & C_1(K) \rightarrow P(*)
\end{array}
$$

$$
\begin{array}{lll}
\multicolumn{3}{l}{\text{Protocol rules :}} \\
\text{A1 :} & P(*)A_0(K) & \rightarrow N_S(enc(K,(x_1,x_2,x_3,0)))A_1(K) \\
\text{A2 :} & N_R(enc(K,(x_1,x_2,x_3,1)))A_1(K) & \rightarrow N_S(enc(K,(x_1,x_2,0,1)))A_2(K) \\
\text{A3 :} & N_R(enc(K,(x_1,x_2,1,0)))A_2(K) & \rightarrow N_S(enc(K,(x_1,0,1,1)))A_3(K) \\
\text{A4 :} & N_R(enc(K,(x_1,1,0,0)))A_3(K) & \rightarrow N_S(enc(K,(0,1,1,1)))A_4(K) \\
\\
\text{B1 :} & N_R(enc(K,(x_1,x_2,x_3,0)))B1_0(K) & \rightarrow N_S(enc(K,(x_1,x_2,x_3,1)))B1_1(K) \\
\text{B2 :} & N_R(enc(K,(x_1,x_2,0,1)))B2_0(K) & \rightarrow N_S(enc(K,(x_1,x_2,1,0)))B2_1(K) \\
\text{B3 :} & N_R(enc(K,(x_1,0,1,1)))B3_0(K) & \rightarrow N_S(enc(K,(x_1,1,0,0)))B3_1(K) \\
\text{B4 :} & N_R(enc(K,(0,1,1,1)))B4_0(K) & \rightarrow N_S(enc(K,(1,0,0,0))B4_1(K) \\
\\
\text{A5 :} & N_R(enc(K,(1,0,0,0)))A_4(K) & \rightarrow N_S(enc(K,(0,0,0,0)))A_5(K) \\
\text{C1 :} & N_R(enc(K,(0,0,0,0)))C_0(K) & \rightarrow N_S(OK)C_1(K) \\
\\
\text{A6 :} & N_R(enc(K,(0,x_1,x_2,x_3)))A_4(K) & \rightarrow N_S(enc(K,(1,1,1,1)))A_5(K) \\
\text{A7 :} & N_R(enc(K,(x_1,1,x_2,x_3)))A_4(K) & \rightarrow N_S(enc(K,(1,1,1,1)))A_5(K) \\
\text{A8 :} & N_R(enc(K,(x_1,x_2,1,x_3)))A_4(K) & \rightarrow N_S(enc(K,(1,1,1,1)))A_5(K) \\
\text{A9 :} & N_R(enc(K,(x_1,x_2,x_3,1)))A_4(K) & \rightarrow N_S(enc(K,(1,1,1,1)))A_5(K) \\
\text{C2 :} & N_R(enc(K,(1,1,1,1)))C_0(K) & \rightarrow N_S(SECRET)C_1(K)
\end{array}
$$

Figure F.25: Protocol theory rules in semi-founded form

Intial set of facts represents key distribution for communication with the server and includes 4 facts representing principals' names. There should be additional 2 facts for role states, one for the server state A_i and another for the principal currently having a session with the server A. Role regeneration theory optimizes the number of facts required by deleting final role states with ERASE rules. Another fact is required for the network predicate. Therefore, a protocol run between A, $B_1, \ldots, B_4$ and C with no intruder involved requires a configuration of at least **11 facts of the size of at least 10**.

Appendix F.1. An exponential attack on the protocol

It is argued in the [15] that this protocol, which was in the restricted well-founded form, is secure against polynomial-time attack and insecure under Dolev-Yao assumptions. There is an attack which requires an exponential number of protocol sessions. Since in a well-founded protocol theory the initial role states are created before protocol execution, this attack would no longer be possible with a balanced well-founded protocol theory and a bounded memory intruder. In a fixed configuration the number of roles would be bounded by the number of facts in the configuration.

In a semi-founded protocol theory there are rules from role regeneration theory which delete final protocol state facts, so the protocols runs with even an exponential number of roles are possible. Although there is only a bounded number of parallel (concurrent) sessions, it is even possible to have an infinite number of roles in a run.

When a Dolev-Yao intruder is present, he can route an initial message $(0, 0, 0, 0)$ encrypted by K from the server A through $2^s - 1$ principals creating an exponential run of the protocol. The value of the encrypted binary number gets increased and finally reaches all 1's which is then sent to C and causes broadcasting of the SECRET.

The intruder only forwards the messages without being able to decrypt them. He uses the FWD rule which does not require any additional intruder's memory. These actions are repeated for each of the 2^s protocol sessions with principals B_i. Finally he sends the last message consisting of all 1's encrypted by K to C who then broadcasts the SECRET. Intruder learns the secret by using the rules REC, DM and then forwards the message to A using USEC and SND rules. For that he needs 2 $R(*)$ facts.

Consequently, the exponential attack requires a configuration of at least **13 facts** of the **size 10**, of which 2 $R(*)$ facts.

Appendix G. Symmetric Key Kerberos 5

Kerberos is a widely deployed protocol, designed to repeatedly authenticate a client to multiple application servers based on a single login. The protocol uses various credentials (tickets), encrypted under a servers key and thus opaque to the client, to authenticate the client to the server. This allows the client to obtain additional credentials or to request service from an application server.

We follow the Kerberos 5 representation from Butler, Cervesato, Jaggard, Scedrov "A Formal Analysis of Some Properties of Kerberos 5 Using MSR". We use the level "A" formalization of Kerberos 5 with mutual authentication which allows the ticket anomaly of the protocol. For simplicity we use t instead of $t_{C,S_{req}}$ timestamp in the last two messages of the protocol shown in the Fig. G.26.

$$
\begin{array}{l}
C \longrightarrow K : C, T, n_1 \\
K \longrightarrow C : C, \{AKey, C\}_{k_T}, \{AKey, n_1, T\}_{k_C} \\
C \longrightarrow T : \{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2 \\
T \longrightarrow C : C, \{SKey, C\}_{k_S}, \{SKey, n_2, S\}_{AKey} \\
C \longrightarrow S : \{SKey, C\}_{k_S}, \{C, t_{c,S_{req}}\}_{SKey} \\
S \longrightarrow C : \{t_{c,S_{req}}\}_{SKey}
\end{array}
$$

Figure G.26: Kerberos 5 Protocol.

A run of Kerberos 5 consists of three successive phases which involve three different servers. It accomplishes a repeated authentification of a client to multiple servers while minimizing the use of the long-term secret key(s) shared between the client and the Kerberos infrastructure. The client C who wishes to authenticate herself to an application server S starts by obtaining a long-term credential, whose use requires her long term (shared) key, and then uses this to obtain short-term credentials for particular servers. In the first phase, C sends a message to the Kerberos Authentication Server (KAS) K requesting a ticket granting ticket (TGT) for use with a particular Ticket Granting Server (TGS) T. K is expected to reply with message consisting of the ticket TGT and an encrypted component containing a fresh authentication key $AKey$ to be shared between C and T. In the second phase, C forwards TGT, along with an authenticator encrypted under $AKey$, to the TGS T as a request for a service ticket for use with the server S. Server T is expected to respond with a message consisting of the service ticket (ST) and an encrypted component containing a fresh service key $SKey$ to be shared between C and S. In the third phase, C forwards ST and a new authenticator encrypted with $SKey$ to S. If all credentials are valid, this application server will authenticate C and provide the service. The last protocol message is an optional acknowledgment message.

A single ticket-granting ticket can be used to obtain several service tickets, possibly from several application servers, while it is valid. Similarly, a single service ticket for the application server S can be used for repeated service from S before it expires. In both cases, a fresh authenticator is required for each use of the ticket.

A semi-founded protocol theory is given in Figure G.27. The additional predicates used in the theory were depicted in Figure B.10.

Initial set of facts consists in facts representing participant's names and servers participating in the protocol, and facts representing secret keys distribution. We assume the secret key of the participant k_C has previously been stored in the key database accessible by the Kerberos Authentication Server K. Similarly we assume the secret key of the Ticket Granting Server T has been stored in the key database accessible by K and the secret key of the Server S has been stored in the key database accessible by the Ticket Granting Server T.
Initial set of facts includes the following 7 facts:

$$\begin{aligned} W = {} & AnnN(C)\ KAS(K)\ TGS(T)\ Server(S) \\ & Guy(C, k_C)\ TGSKey(T, k_T)\ ServerKey(S, k_S)\ . \end{aligned}$$

There should be additional 4 facts for role state predicates and another fact for the network predicate.
Rules marked with $\rightarrow_{clock}, \rightarrow_{constraint_K}, \rightarrow_{constraint_T}$ and $\rightarrow_{constraint_S}$ represent constraints related to timestamps and to validity of relevant Kerberos messages. They are determined by an external process and we represent them with separate rules:

$$\begin{aligned} \text{constraint}_K : \quad & P(*) \rightarrow Valid_K(C, T, n_1) \\ \text{constraint}_T : \quad & P(*) \rightarrow Valid_T(C, S, n_2) \\ \text{constraint}_S : \quad & P(*) \rightarrow Valid_S(C, t) \\ \text{clock} : \quad & P(*) \rightarrow Clock_C(t) \end{aligned}$$

Additional facts representing memory, clock and validity constraints, *i.e.*$Auth$, $Service$, $DoneMut_C$, Mem_S, $Clock$, $Valid_K$, $Valid_T$, $Valid_S$, require 3 facts (not all are persistent so we don't need all 8 facts).
Therefore, a protocol run between the client C and Kerberos servers K,T and S with no intruder involved requires a configuration of at least **15 facts** of the **size of at least 16**.

Role Regeneration Theory :

ROLC : $Guy(G, k_G)\ AnnN(G)\ P(*) \rightarrow Guy(G, k_G)\ AnnN(G)\ C_0(C)$
ROLK : $KAS(K)\ P(*) \rightarrow KAS(K)\ K_0(K)$
ROLT : $TGS(T)\ P(*) \rightarrow TGS(T)\ T_0(T)$
ROLS : $Server(S)\ P(*) \rightarrow Server(S)\ S_0(S)$
ERASEC : $C_4(C, S, SKey, t, Y) \rightarrow P(*)$
ERASEK : $K_1(K) \rightarrow P(*)$
ERASET : $T_1(T) \rightarrow P(*)$
ERASES : $S_1(S) \rightarrow P(*)$

Protocol Theories $\mathcal{C}$, $\mathcal{K}$, $\mathcal{T}$ and $\mathcal{S}$:

C1 : $C_0(C)\ TGS(T)\ P(*) \rightarrow \exists n_1.C_1(C, T, n_1)\ TGS(T)\ N_S(\langle C, \langle T, n_1\rangle\rangle)$
C2 : $C_1(C, T, n_1)\ Server(S)\ N_R(\langle C, \langle X, enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle)\ P(*)$
$\rightarrow \exists n_2.C_2(C, T, S, AKey, n_2)\ Server(S)\ Auth(X, T, AKey)$
$N_S(\langle X, \langle enc(AKey, C), \langle C, \langle S, n_2\rangle\rangle\rangle\rangle)$
C3 : $C_2(C, T, S, AKey, n_2)\ Clock_C(t)$
$N_R(\langle C, \langle Y, enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle)$
$\rightarrow C_3(C, S, SKey, t, Y)\ N_S(\langle Y, enc(SKey, \langle C, t\rangle)\rangle)\ Service(Y, S, SKey)$
C4 : $C_3(C, S, SKey, t, Y)\ N_R(enc(SKey, t))$
$\rightarrow C_4(C, S, SKey, t, Y)\ DoneMut_C(S, SKey)$

K1 : $K_0(K)\ Guy(C, k_C)\ TGSKey(T, k_T)\ N_R((\langle C, \langle T, n_1\rangle\rangle))\ Valid_K(C, T, n_1)$
$\rightarrow \exists AKey.K_1(K)\ Guy(C, k_C)\ TGSKey(T, k_T)\ P(*)$
$N_S(\langle C, \langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle)$
T1 : $T_0(T)\ TGSKey(T, k_T)\ ServerKey(S, k_S)\ Valid_T(C, S, n_2)$
$N_R(\langle enc(k_T, \langle AKey, C\rangle), \langle enc(AKey, C), \langle C, \langle S, n_2\rangle\rangle\rangle\rangle)$
$\rightarrow \exists SKey.T_1(T)\ TGSKey(T, k_T)\ SerevrKey(S, k_S)\ P(*)$
$N_S(\langle C, \langle enc(k_S, \langle SKey, C\rangle), enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle)$
S1 : $S_0(S)\ ServerKey(S, k_S)\ Valid_S(C, t)$
$N_R(\langle enc(k_S, \langle SKey, C\rangle), enc(SKey, \langle C, t\rangle)\rangle)$
$\rightarrow S_1(S)\ ServerKey(S, k_S)\ N_S(enc(SKey, t))\ Mem_S(C, SKey, t)$

Figure G.27: Semi-founded protocol theory for the Kerberos 5 Protocol.

The trace representing the normal protocol run is given below:

$$
\begin{array}{l}
W\ C_0(C)\ K_0(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*)P(*) \rightarrow_{C1} \\
W\ C_1(C, T, n_1)\ K_0(K)\ T_0(T)\ S_0(S)\ N(\langle C, \langle T, n_1\rangle\rangle) \\
\qquad P(*)P(*)P(*) \rightarrow_{constraint_K} \\
W\ C_1(C, T, n_1)\ K_0(K)\ T_0(T)\ S_0(S) \\
\qquad N(\langle C, \langle T, n_1\rangle\rangle)\ Valid_K(C, T, n_1)P(*)P(*) \rightarrow_{K1} \\
W\ C_1(C, T, n_1)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*) \\
\qquad N(\langle C, \langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle) \rightarrow_{C2} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad N(\langle C, \langle\langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{constraint_T} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Valid_T(C, S, n_2) \\
\qquad N(\langle C, \langle\langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{T1} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ P(*)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad N(\langle C, \langle enc(k_S, \langle SKey, C\rangle), enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle) \rightarrow_{clock} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ Clock_C(t)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad N(\langle C, \langle enc(k_S, \langle SKey, C\rangle), enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle) \rightarrow_{C3} \\
W\ C_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(K)\ T_1(T)\ S_0(S)\ P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad Service(enc(k_S, \langle SKey, C\rangle, S, SKey) \\
\qquad N(\langle enc(k_S, \langle SKey, C\rangle, enc(SKey, \langle C, t\rangle)\rangle) \rightarrow_{constraints_S} \\
W\ C_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(K)\ T_1(T)\ S_0(S) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad Service(enc(k_S, \langle SKey, C\rangle, S, SKey)\ Valid_S(C, t) \\
\qquad N(\langle enc(k_S, \langle SKey, C\rangle, enc(SKey, \langle C, t\rangle)\rangle) \rightarrow_{S1} \\
W\ C_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(K)\ T_1(T)\ S_1(S) \\
\qquad Service(enc(k_S, \langle SKey, C\rangle, S, SKey)\ Mem_S(C, SKey, t) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ N(enc(SKey, t)) \rightarrow_{C4} \\
W\ C_4(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(K)\ T_1(T)\ S_1(S) \\
\qquad Service(enc(k_S, \langle SKey, C\rangle, S, SKey)\ Mem_S(C, SKey, t) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ DoneMut_C(S, SKey)
\end{array}
$$

Appendix G.1. Ticket anomaly in Kerberos 5 protocol

The informal description of the ticket anomaly in Kerberos 5 protocol is given in Figure G.28. Intruder intercepts the message from K and replaces the ticket with a generic (dummy) message X and stores the actual ticket in his memory. C cannot detect this as he aspects the opaque sub-message representing the ticket therefore just forwards the received meaningless X. Intruder intercepts this message and replaces X with the original ticket from K. He forwards the well-formed message to server T and rest of the protocol proceeds as normal.

$$
\begin{array}{l}
C \longrightarrow K : C, T, n_1 \\
K \longrightarrow I(C) : C, \{AKey, C\}_{k_T}, \{AKey, n_1, T\}_{k_C} \\
I(K) \longrightarrow C : C, X, \{AKey, n_1, T\}_{k_C} \\
C \longrightarrow I(T) : X, \{C\}_{AKey}, C, S, n_2 \\
I(C) \longrightarrow T : \{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2 \\
T \longrightarrow C : C, \{SKey, C\}_{k_S}, \{SKey, n_2, S\}_{AKey} \\
C \longrightarrow S : \{SKey, C\}_{k_S}, \{C, t_{c,S_{req}}\}_{SKey} \\
S \longrightarrow C : \{t_{c,S_{req}}\}_{SKey}
\end{array}
$$

Figure G.28: Ticket anomaly in Kerberos 5 protocol

As the result of the intruder's actions the server T has granted the client C a ticket for the server S even though C has never received nor sent a valid second Kerberos 5 message to T (C only thinks he has). Furthermore, since Kerberos 5 allows multiple ticket use, subsequent attempts from C to get the ticket for the server S with a dummy ticket granting ticket X will fail for reasons unknown to C.

In order to perform this attack intruder should be able to generate a generic message of the type $msgaux < msg$ representing a "false ticket". Later on he should store this type of data in a separate memory predicate M_m. Therefore we use rules GENM, LRNM and USEM from the intruder theory.

$$
\begin{array}{l}
\text{GENM} : R(*) \to \exists m. M_m(m) \\
\text{LRNM} : D(m) \to M_m(m) \\
\text{USEM} : M_m(m) R(*) \to M_m(m)\; C(m)
\end{array}
$$

As in the normal run with no intruder present, initial set of 7 facts is:

$$
\begin{array}{rl}
W = & AnnN(C)\; KAS(K)\; TGS(T)\; Server(S) \\
& Guy(C, k_C)\; TGSKey(T, k_T)\; ServerKey(S, k_S)\;.
\end{array}
$$

A trace representing the anomaly is shown below.

$$
\begin{array}{l}
WC_0(C)K_0(k_C,k_T)T_0(k_S)S_0(S) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*)P(*) \to_{C1} \\
WC_1(C,T,n_1)K_0(k_C,k_T)T_0(k_S)S_0(S)\ N_S(\langle C,\langle T,n_1\rangle\rangle) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \to
\end{array}
$$

Intruder forwards the message to the server K.

$$
\begin{array}{l}
\to_{FWD} \\
WC_1(C,T,n_1)K_0(k_C,k_T)T_0(k_S)S_0(S)\ N_R(\langle C,\langle T,n_1\rangle\rangle) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \to_{constraint_K} \\
WC_1(C,T,n_1)K_0(k_C,k_T)T_0(k_S)S_0(S)\ Valid_K(C,T,n_1) \\
\quad N_S(\langle C,\langle T,n_1\rangle\rangle)\ R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*) \to_{K1} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \\
\quad N_S(\langle C,\langle enc(k_T,\langle AKey,C\rangle),enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle) \to
\end{array}
$$

Intruder intercepts the reply from the server K and digests parts of its contents.

$$
\begin{array}{l}
\to_{REC} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*)P(*) \\
\quad D(\langle C,\langle enc(k_T,\langle AKey,C\rangle),enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle) \to_{DCMP} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*)P(*) \\
\quad D(C)D(\langle enc(k_T,\langle AKey,C\rangle),enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle) \to_{DCMP} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad R(*)R(*)R(*)R(*)P(*)P(*)P(*)P(*) \\
\quad D(C)D(enc(k_T,\langle AKey,C\rangle))\ D(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \to_{LRNG} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad R(*)R(*)R(*)R(*)P(*)P(*)P(*)P(*) \\
\quad M_g(C)D(enc(k_T,\langle AKey,C\rangle))D(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \to
\end{array}
$$

Intruder bins the part of the message he does not need since he will replace it later with a fresh generic message that he generates.

$$
\begin{array}{l}
\to_{DM^2} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_g(C) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle))\ M_s(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad R(*)R(*)R(*)P(*)P(*)P(*)P(*) \to
\end{array}
$$

$$
\begin{array}{l}
\rightarrow_{GENM} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_g(C)M_g(T)M_c(X) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle))\ M_s(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad R(*)R(*)P(*)P(*)P(*)P(*) \rightarrow_{USES} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_g(C)M_g(T)M_c(X) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle))\ M_s(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad C(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad P(*)P(*)P(*)P(*)P(*) \rightarrow_{USEM} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_g(C)M_g(T)M_c(X) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle))\ M_s(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad C(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle))\ C(X) \\
\quad P(*)P(*)P(*)P(*) \rightarrow_{COMP} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_g(C)M_g(T)M_c(X) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle))\ M_s(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad C(\langle X,enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle) \\
\quad P(*)R(*)P(*)P(*)P(*) \rightarrow_{USEG} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_g(C)M_g(T)M_c(X) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle))\ M_s(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad C(\langle X,enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle)\ C(C) \\
\quad P(*)P(*)P(*)P(*) \rightarrow_{COMP} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_g(C)M_g(T)M_c(X) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle))\ M_s(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad C(\langle C,\langle X,enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle) \\
\quad R(*)P(*)P(*)P(*)P(*) \rightarrow_{SND} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_g(C)M_g(T)M_c(X) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle))\ M_s(enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)) \\
\quad N_R(\langle C,\langle X,enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle) \\
\quad R(*)R(*)P(*)P(*)P(*) \rightarrow
\end{array}
$$

Intruder uses memory maintenance rules to free the memory of unnecessary facts including the $B(*)$ facts. In order to perform the attack ne needs to keep the ticket granting ticket in his memory.

$$
\begin{array}{l}
\rightarrow_{DEL4} \\
WC_1(C,T,n_1)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)M_s(enc(k_T,\langle AKey,C\rangle)) \\
\quad N_R(\langle C,\langle X,enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \rightarrow
\end{array}
$$

Client C does not notice the faulty message since he expects to receive an opaque submessage representing a ticket granting ticket, therefore re replies as if the message was a valid message from K.

$$
\begin{array}{l}
\to_{C2} \\
WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad N_S(\langle X,\langle enc(AKey,C),\langle C,\langle S,n_2\rangle\rangle\rangle\rangle)\ Auth(X,T,AKey) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle)) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*) \to
\end{array}
$$

Intruder intercepts the message and needs to replace the generic message X with the original ticket granting ticket. We use the notation $X = enc(k_T,\langle AKey,C\rangle)$.

$$
\begin{array}{l}
\to_{REC} \\
WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad D(\langle X,\langle enc(AKey,C),\langle C,\langle S,n_2\rangle\rangle\rangle\rangle)\ Auth(X,T,AKey) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle)) \\
\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \to_{DCMP} \\
WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad D(X)\ D(\langle enc(AKey,C),\langle C,\langle S,n_2\rangle\rangle\rangle)\ Auth(X,T,AKey) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle)) \\
\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \to_{DELD} \\
WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad B*(*)\ D(\langle enc(AKey,C),\langle C,\langle S,n_2\rangle\rangle\rangle)\ Auth(X,T,AKey) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle)) \\
\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \to_{DCMPB} \\
WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad D(enc(AKey,C))\ D(\langle C,\langle S,n_2\rangle\rangle)\ Auth(X,T,AKey) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle)) \\
\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \to_{DM^2} \\
WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad M_s(enc(AKey,C))\ M_s(\langle C,\langle S,n_2\rangle\rangle)\ Auth(X,T,AKey) \\
\quad M_s(enc(k_T,\langle AKey,C\rangle)) \\
\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \to
\end{array}
$$

For the composition of the message intruder needs 2 additional $R(*)$ facts.

$$
\begin{aligned}
&\rightarrow_{(USES^2,COMP,USES,COMP)}\\
&WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)\\
&\quad M_s(enc(AKey,C))\ M_s(\langle C,\langle S,n_2\rangle\rangle)\ Auth(X,T,AKey)\\
&\quad C(\langle enc(k_T,\langle AKey,C\rangle),\langle enc(AKey,C),\langle C,\langle S,n_2\rangle\rangle\rangle\rangle)\\
&\quad R(*)R(*)R(*)R(*)P(*)P(*)P(*) \rightarrow_{SND}\\
&WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)\\
&\quad M_s(enc(AKey,C))\ M_s(\langle C,\langle S,n_2\rangle\rangle)\ Auth(X,T,AKey)\\
&\quad N_R(\langle enc(k_T,\langle AKey,C\rangle),\langle enc(AKey,C),\langle C,\langle S,n_2\rangle\rangle\rangle\rangle)\\
&\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*) \rightarrow_{DEL_2}\\
&WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)\\
&\quad Auth(X,T,AKey)\\
&\quad N_R(\langle enc(k_T,\langle AKey,C\rangle),\langle enc(AKey,C),\langle C,\langle S,n_2\rangle\rangle\rangle\rangle)\\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*) \rightarrow
\end{aligned}
$$

In the rest of the protocol intruder only forwards the messages using FWD rule.

$$
\begin{aligned}
&\rightarrow_{constraint_T}\\
&WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)P(*)\\
&\quad Auth(X,T,AKey)\ Valid_T(C,S,n_2)\\
&\quad N_R(\langle, enc(k_T,\langle AKey,C\rangle),\langle enc(AKey,C),\langle C,\langle S,n_2\rangle\rangle\rangle\rangle)\\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow
\end{aligned}
$$

$$
\begin{aligned}
\rightarrow_{T_1}\ &WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)P(*)P(*)\\
&N_S(\langle C,\langle enc(k_S,\langle SKey,C\rangle),enc(AKey,\langle SKey,\langle n_2,S\rangle\rangle)\rangle\rangle)\\
&Auth(X,T,AKey)R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow
\end{aligned}
$$

Intruder only forwards the remaining messages since it does not help him in any way to keep any data from the message in the memory.
We use the notation $Y = enc(k_S,\langle SKey,C\rangle)$.

$$
\begin{aligned}
&\rightarrow_{FWD}\\
&WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)P(*)P(*)\\
&\quad Auth(X,T,AKey)\\
&\quad N_R(\langle C,\langle enc(k_S,\langle SKey,C\rangle),enc(AKey,\langle SKey,\langle n_2,S\rangle\rangle)\rangle\rangle)\\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow_{clock}\\
&WC_2(C,T,S,AKey,n_2)K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)P(*)\\
&\quad Auth(X,T,AKey)\ Clock_C(t)\\
&\quad N_R(\langle C,\langle enc(k_S,\langle SKey,C\rangle),enc(AKey,\langle SKey,\langle n_2,S\rangle\rangle)\rangle\rangle)\\
&\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow
\end{aligned}
$$

$$
\begin{array}{l}
\rightarrow_{C3} \\
WC_3(C,S,SKey,t,Y)\ K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)P(*) \\
\quad\quad Auth(X,T,AKey)\ Service(Y,S,SKey) \\
\quad\quad N_S(\langle Y, enc(SKey,\langle C,t\rangle)\rangle) \\
\quad\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow_{FWD} \\
WC_3(C,S,SKey,t,Y)\ K_1(k_C,k_T,AKey)T_0(k_S)S_0(S)P(*) \\
\quad\quad Auth(X,T,AKey)\ Service(Y,S,SKey) \\
\quad\quad N_R(\langle Y, enc(SKey,\langle C,t\rangle)\rangle) \\
\quad\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow_{constraint_S} \\
WC_3(C,S,SKey,t,Y)\ K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad\quad Auth(X,T,AKey)\ Service(Y,S,SKey)\ Valid_S(C,t) \\
\quad\quad N_R(\langle Y, enc(SKey,\langle C,t\rangle)\rangle) \\
\quad\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow_{S1} \\
WC_3(C,S,SKey,t,Y)\ K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad\quad Auth(X,T,AKey)\ Service(Y,S,SKey)\ Mem_S(C,SKey,t) \\
\quad\quad N_S(enc(SKey,t)) \\
\quad\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow_{FWD} \\
WC_3(C,S,SKey,t,Y)\ K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad\quad Auth(X,T,AKey)\ Service(Y,S,SKey)\ Mem_S(C,SKey,t) \\
\quad\quad N_R(enc(SKey,t)) \\
\quad\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow_{C4} \\
WC_4(C,S,SKey,t,Y)\ K_1(k_C,k_T,AKey)T_0(k_S)S_0(S) \\
\quad\quad Auth(X,T,AKey)\ \ Service(Y,S,SKey)\ Mem_S(C,SKey,t) \\
\quad\quad DoneMut_C(S,SKey) \\
\quad\quad R(*)R(*)R(*)R(*)R(*)R(*)R(*)
\end{array}
$$

With respect to memory, it does help the intruder to be "clever". The attack requires a configuration of at least **22 facts** (15 for the protocol and additional 7 facts for the intruder) of the **size 16**.

Appendix G.2. Replay anomaly in Kerberos 5 protocol

”A” level formalization of Kerberos 5 does not include some nonces and timestamps of the protocol, so it precludes detection of replayed messages. Request messages that client sends to servers can therefore be stored in intruder’s memory when he intercepts them. Later on he can put them on the network as additional requests. If the original requests were accepted by the servers, so may be the replayed ones as well. In that case the server generates fresh credentials based on replayed requests. Differently than in the case of ticket anomaly, fresh credentials are granted.

$$
\begin{array}{rl}
C \longrightarrow K: & C, T, n_1 \\
K \longrightarrow C: & C, \{AKey, C\}_{k_T}, \{AKey, n_1, T\}_{k_C} \\
C \longrightarrow G: & \{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_2 \\
G \longrightarrow C: & C, \{SKey, C\}_{k_S}, \{SKey, n_2, S\}_{AKey} \\
C \longrightarrow I(S): & \{SKey, C\}_{k_S}, \{C, t_{c,S_{req}}\}_{SKey} \\
I(C) \longrightarrow S: & \{SKey, C\}_{k_S}, \{C, t_{c,S_{req}}\}_{SKey} \\
S \longrightarrow C: & \{t_{c,S_{req}}\}_{SKey} \\
I(C) \longrightarrow S: & \{SKey, C\}_{k_S}, \{C, t_{c,S_{req}}\}_{SKey} \\
S \longrightarrow I(C): & \{t_{c,S_{req}}\}_{SKey}
\end{array}
$$

Figure G.29: Replay anomaly of Kerberos 5 Protocol

We will model the replay of the third request message from the protocol, as shown in Figure G.29.

Intruder basically observes the protocol run remembering the request message to the Server. He only digests the network predicates, *i.e.*transforms the N_S to N_R predicate. Some messages are only forwarded with all of the data learnt from them deleted, while the data from the request message is kept in intruder’s memory for later replay. Differently from ticket anomaly, intruder does not generate any fresh data.

As in the normal run with no intruder present, initial set of 7 facts is:

$$
\begin{aligned}
W = {} & AnnN(C)\ KAS(K)\ TGS(T)\ Server(S) \\
& Guy(C, k_C)\ TGSKey(T, k_T)\ ServerKey(S, k_S)\ .
\end{aligned}
$$

This attack requires a configuration of at least **20 facts** (16 for the protocol and additional 4 facts for the intruder) of the **size 16**, as shows the trace of the anomaly given below.

$$
\begin{array}{l}
W C_0(C) K_0(k_C, k_T) T_0(k_S) S_0()\ P(*)P(*)P(*)P(*)P(*) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow_{C1} \\
W\ C_1(C, T, n_1)\ K_0(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*)P(*)\ N_S(\langle C, \langle T, n_1\rangle\rangle) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow
\end{array}
$$

Intruder simply forwards the messages he's not interested in.

$$
\begin{array}{l}
\rightarrow_{FWD} \\
W\ C_1(C, T, n_1)\ K_0(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*)P(*)\ N_R(\langle C, \langle T, n_1\rangle\rangle) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow_{constraint_K} \\
W\ C_1(C, T, n_1)\ K_0(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*) \\
\qquad N_R(\langle C, \langle T, n_1\rangle\rangle)\ Valid_K(C, T, n_1) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow_{K1} \\
W\ C_1(C, T, n_1)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*)P(*) \\
\qquad N_S(\langle C, \langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow
\end{array}
$$

Intruder again forwards the message.

$$
\begin{array}{l}
\rightarrow_{FWD} \\
W\ C_1(C, T, n_1)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*)P(*) \\
\qquad N_R(\langle C, \langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow_{C2} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad N_R(\langle C, \langle\langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow_{constraint_T} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Valid_T(C, S, n_2) \\
\qquad N_R(\langle C, \langle\langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow_{T1} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ P(*)P(*)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad N_S(\langle C, \langle enc(k_S, \langle SKey, C\rangle), enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle) \\
\qquad R(*)R(*)R(*)R(*) \rightarrow
\end{array}
$$

Intruder only forwards the message.

$$
\begin{array}{l}
\rightarrow_{FWD} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ P(*)P(*)P(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey) \\
\quad N_R(\langle C,\langle enc(k_S,\langle SKey,C\rangle),enc(AKey,\langle SKey,\langle n_2,S\rangle\rangle)\rangle\rangle) \\
\quad R(*)R(*)R(*)R(*) \rightarrow_{clock} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ Clock_C(t)\ P(*)P(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey) \\
\quad N(\langle C,\langle enc(k_S,\langle SKey,C\rangle),enc(AKey,\langle SKey,\langle n_2,S\rangle\rangle)\rangle\rangle) \\
\quad R(*)R(*)R(*)R(*) \rightarrow_{C3} \\
WC_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(k_C,k_T,AKey)T_0(k_S)S_0() \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Service(enc(k_S,\langle SKey,C\rangle),S,SKey) \\
\quad N_S(\langle enc(k_S,\langle SKey,C\rangle),enc(SKey,\langle C,t\rangle)\rangle) \\
\quad R(*)R(*)R(*)R(*)P(*)P(*) \rightarrow
\end{array}
$$

Intruder needs data contained in this message therefore he intercepts the message and stores its data.

$$
\begin{array}{l}
\rightarrow_{REC} \\
WC_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(k_C,k_T,AKey)T_0(k_S)S_0() \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Service(enc(k_S,\langle SKey,C\rangle),S,SKey) \\
\quad D(\langle enc(k_S,\langle SKey,C\rangle),enc(SKey,\langle C,t\rangle)\rangle) \\
\quad R(*)R(*)R(*)P(*)P(*)P(*) \rightarrow_{DCMP} \\
WC_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(k_C,k_T,AKey)T_0(k_S)S_0() \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Service(enc(k_S,\langle SKey,C\rangle),S,SKey) \\
\quad D(enc(k_S,\langle SKey,C\rangle))\ D(enc(SKey,\langle C,t\rangle)) \\
\quad R(*)R(*)P(*)P(*)P(*) \rightarrow_{DM^2} \\
WC_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(k_C,k_T,AKey)T_0(k_S)S_0() \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Service(enc(k_S,\langle SKey,C\rangle),S,SKey) \\
\quad M_s(enc(k_S,\langle SKey,C\rangle))\ M_s(enc(SKey,\langle C,t\rangle)) \\
\quad R(*)R(*)P(*)P(*)P(*) \rightarrow
\end{array}
$$

Intruder starts composing the message.

$$
\begin{array}{l}
\rightarrow_{USES^2} \\
WC_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(k_C,k_T,AKey)T_0(k_S)S_0() \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Service(enc(k_S,\langle SKey,C\rangle),S,SKey) \\
\quad M_s(enc(k_S,\langle SKey,C\rangle))\ C(enc(k_S,\langle SKey,C\rangle)) \\
\quad M_s(enc(SKey,\langle C,t\rangle)\ C(enc(SKey,\langle C,t\rangle)) \\
\quad P(*)P(*)P(*) \rightarrow
\end{array}
$$

Notice that there are no $R(*)$ facts in the configuration.

$$
\begin{array}{l}
\rightarrow_{COMP} \\
WC_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(k_C, k_T, AKey) T_0(k_S) S_0() \\
\quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Service(enc(k_S, \langle SKey, C\rangle), S, SKey) \\
\quad M_s(enc(k_S, \langle SKey, C\rangle))\ M_s(enc(SKey, \langle C, t\rangle)) \\
\quad C(enc(k_S, \langle SKey, C\rangle)), enc(SKey, \langle C, t\rangle)) \\
\quad R(*) P(*) P(*) P(*) \rightarrow_{SND} \\
WC_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(k_C, k_T, AKey) T_0(k_S) S_0() \\
\quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Service(enc(k_S, \langle SKey, C\rangle), S, SKey) \\
\quad M_s(enc(k_S, \langle SKey, C\rangle))\ M_s(enc(SKey, \langle C, t\rangle)) \\
\quad N_R(enc(k_S, \langle SKey, C\rangle)), enc(SKey, \langle C, t\rangle)) \\
\quad R(*) R(*) P(*) P(*) \rightarrow_{constraints_S} \\
WC_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(k_C, k_T, AKey) T_0(k_S) S_0() \\
\quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Service(enc(k_S, \langle SKey, C\rangle), S, SKey) \\
\quad Valid_S(C, t)\ N_R(\langle enc(k_S, \langle SKey, C\rangle), enc(SKey, \langle C, t\rangle)\rangle) \\
\quad M_s(enc(k_S, \langle SKey, C\rangle))\ M_s(enc(SKey, \langle C, t\rangle)) \\
\quad R(*) R(*) P(*) \rightarrow_{S_1} \\
WC_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(k_C, k_T, AKey) T_0(k_S) S_1() \\
\quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Mem_S(C, SKey, t) \\
\quad Service(enc(k_S, \langle SKey, C\rangle), S, SKey)\ N_S(enc(SKey, t)) \\
\quad M_s(enc(k_S, \langle SKey, C\rangle))\ M_s(enc(SKey, \langle C, t\rangle)) \\
\quad R(*) R(*) P(*) \rightarrow
\end{array}
$$

Again intruder only forwards the message.

$$
\begin{array}{l}
\rightarrow_{FWD} \\
WC_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(k_C, k_T, AKey) T_0(k_S) S_1() \\
\quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Mem_S(C, SKey, t) \\
\quad Service(enc(k_S, \langle SKey, C\rangle), S, SKey)\ N_R(enc(SKey, t) \\
\quad M_s(enc(k_S, \langle SKey, C\rangle))\ M_s(enc(SKey, \langle C, t\rangle)) \\
\quad R(*) R(*) P(*) \rightarrow_{C4} \\
WC_4(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(k_C, k_T, AKey) T_0(k_S) S_1() \\
\quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Mem_S(C, SKey, t) \\
\quad Service(enc(k_S, \langle SKey, C\rangle), S, SKey)\ DoneMut_C(S, SKey) \\
\quad M_s(enc(k_S, \langle SKey, C\rangle))\ M_s(enc(SKey, \langle C, t\rangle)) \\
\quad R(*) R(*) P(*) \rightarrow
\end{array}
$$

After this run has completed, intruder replays the request to the Server S.

Role regeneration theory rules ROLS and ERASES allow another session with the Server.

$$
\begin{array}{l}
\rightarrow_{(ERASES,ROLS,USES^2,COMP,SND)} \\
WC_4(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(k_C,k_T,AKey)T_0(k_S)S_0() \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Mem_S(C,SKey,t) \\
\quad Service(enc(k_S,\langle SKey,C\rangle),S,SKey)\ DoneMut_C(S,SKey) \\
\quad M_s(enc(k_S,\langle SKey,C\rangle))\ M_s(enc(SKey,\langle C,t\rangle)) \\
\quad N_R(\langle enc(k_S,\langle SKey,C\rangle),enc(SKey,\langle C,t\rangle)\rangle) \\
\quad R(*)R(*) \rightarrow_{S1} \\
WC_4(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(k_C,k_T,AKey)T_0(k_S)S_1() \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Mem_S(C,SKey,t) \\
\quad Service(enc(k_S,\langle SKey,C\rangle),S,SKey)\ DoneMut_C(S,SKey) \\
\quad M_s(enc(k_S,\langle SKey,C\rangle))\ M_s(enc(SKey,\langle C,t\rangle))\ N_S(enc(SKey,t)) \\
\quad R(*)R(*)
\end{array}
$$

Appendix H. Public Key extension of Kerberos 5 - PKINIT

The Public Key extension of Kerberos 5 differs from the symmetric version of Kerberos 5 in the initial round between the client and the KAS. Public key encryption is used instead of a shared key between the client and the KAS.

In the PKINIT the client C and the KAS possess independent public and secret key pairs, (pk_C, sk_C) and (pk_K, sk_K), respectively. Certificate sets $Cert_C$ and $Cert_K$ testify the binding of the principal and her public key. The rest of the protocol remains unchanged, see Fig. H.30, where for simplicity we use t instead of $t_{C,S_{req}}$ timestamp in the last two messages of the protocol. We keep a similar level of abstraction as in the previous section on Kerberos 5.

A semi-founded protocol theory for the PKINIT protocol is given in Figure H.31.

$$
\begin{array}{l}
C \longrightarrow K : Cert_C, \{t_C, n_2\}_{sk_C}, C, T, n_1 \\
K \longrightarrow C : \{Cert_K, \{k, n_2\}_{sk_K}\}_{pk_C}, C, \{AKey, C\}_{k_T}, \{AKey, n_1, t_K, T\}_k \\
C \longrightarrow T : \{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_3 \\
T \longrightarrow C : C, \{SKey, C\}_{k_S}, \{SKey, n_3, S\}_{AKey} \\
C \longrightarrow S : \{SKey, C\}_{k_S}, \{C, t_{c,S_{req}}\}_{SKey} \\
S \longrightarrow C : \{t_{c,S_{req}}\}_{SKey}
\end{array}
$$

Figure H.30: PKINIT Protocol.

We show that a PKINIT protocol run between the client C and Kerberos servers K,T and S with no intruder involved requires a configuration of at least **18 facts** of the **size of at least 28**.

Initial set of facts consists of facts representing participant's names and servers participating in the protocol, and facts representing secret keys and public/private key distribution. We assume the secret key of the Ticket Granting Server T has been stored in the key database accessible by K and the secret key of the Server S has been stored in the key database accessible by the Ticket Granting Server T. Initial set of facts has 10 facts:

$$
\begin{aligned}
W = \ & Client(C, pk_C)\ KP(pk_C, sk_C)\ AnnK(pk_C) \\
& KAS(K)\ KP(pk_K, sk_K)\ AnnK(pk_K) \\
& TGS(T)\ TGSKey(T, k_T) \\
& Server(S)\ ServerKey(S, k_S)\ .
\end{aligned}
\tag{H.1}
$$

Role Regeneration Theory :

ROLC : $Client(C, pk_C)\ P(*) \rightarrow Client(C, pk_C)\ C_0(C)$
ROLK : $KAS(K)\ P(*) \rightarrow KAS(K)\ K_0(K)$
ROLT : $TGS(T)\ P(*) \rightarrow TGS(T)\ T_0(T)$
ROLS : $Server(S)\ P(*) \rightarrow Server(S)\ S_0(S)$
ERASEC : $C_4(C, S, SKey, t, Y) \rightarrow P(*)$
ERASEK : $K_1(K) \rightarrow P(*)$
ERASET : $T_1(T) \rightarrow P(*)$
ERASES : $S_1(S) \rightarrow P(*)$

Protocol Theories $\mathcal{C}$, $\mathcal{K}$, $\mathcal{T}$ and $\mathcal{S}$:

C1 : $C_0(C)\ TGS(T)\ Clock_C(t_C) \rightarrow \exists n_1.n_2.C_1(C, T, n_1, n_2, t_C)\ TGS(T)$
$\quad N_S(\langle Cert_C, \langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle\rangle)$
C2 : $C_1(C, T, n_1, n_2, t_C)\ Server(S)\ P(*)$
$\quad N_S(\langle enc(pk_C, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle),$
$\quad\quad \langle C, \langle X, enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle\rangle)$
$\quad \rightarrow \exists n_3.C_2(C, T, S, AKey, n_3)\ Server(S)\ Auth(X, T, AKey)$
$\quad\quad N_S(\langle X, \langle enc(AKey, C), \langle C, \langle S, n_3\rangle\rangle\rangle\rangle)$
C3 : $C_2(C, T, S, AKey, n_3)\ N_R(\langle C, \langle Y, enc(AKey, \langle SKey, \langle n_3, S\rangle\rangle)\rangle\rangle)\ Clock_C(t)$
$\quad \rightarrow C_3(C, S, SKey, t, Y)\ N_S(\langle Y, enc(SKey, \langle C, t\rangle)\rangle)\ Service(Y, S, SKey)$
C4 : $C_3(C, S, SKey, t, Y)\ N_R(enc(SKey, t))$
$\quad \rightarrow C_4(C, S, SKey, t, Y)\ DoneMut_C(S, SKey)$

K1 : $K_0(K)\ Client(C, pk_C)\ TGSKey(T, k_T)\ Valid_K(C, T, n_1)\ Clock_K(t_K)$
$\quad N_R(\langle Cert_C, \langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle\rangle)$
$\quad \rightarrow \exists k.AKey.K_1(K)\ Client(C, pk_C)\ TGSKey(T, k_T)\ P(*)P(*)$
$\quad\quad N_S(\langle enc(pk_C, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle),$
$\quad\quad\quad \langle C, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle\rangle)$
T1 : $T_0(T)\ TGSKey(T, k_T)\ ServerKey(S, k_S)\ Valid_T(C, S, n_2)$
$\quad N_R(\langle enc(k_T, \langle AKey, C\rangle), \langle enc(AKey, C), \langle C, \langle S, n_2\rangle\rangle\rangle\rangle)$
$\quad \rightarrow \exists SKey.T_1(T)\ TGSKey(T, k_T)\ SerevrKey(S, k_S)\ P(*)$
$\quad\quad N_S(\langle C, \langle enc(k_S, \langle SKey, C\rangle), enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle)$
S1 : $S_0(S)\ ServerKey(S, k_S)\ Valid_S(C, t)$
$\quad N_R(\langle enc(k_S, \langle SKey, C\rangle), enc(SKey, \langle C, t\rangle)\rangle)$
$\quad \rightarrow S_1(S)\ ServerKey(S, k_S)\ N_S(enc(SKey, t))\ Mem_S(C, SKey, t)$

Figure H.31: Semi-founded protocol theory for the PKINIT

Same as with the symmetric Kerberos 5, rules that are marked with $\rightarrow_{clock_C}$, $\rightarrow_{clock_K}$, $\rightarrow_{constraint_K}$, $\rightarrow_{constraint_T}$ and $\rightarrow_{constraint_S}$ represent constraints related to timestamps and to validity of relevant Kerberos messages. They are determined by an external process and we represent them with separate rules:

$$\begin{array}{rl}
\text{constraint}_K: & P(*) \rightarrow Valid_K(C, T, n_1) \\
\text{constraint}_T: & P(*) \rightarrow Valid_T(C, S, n_2) \\
\text{constraint}_S: & P(*) \rightarrow Valid_S(C, t) \\
\text{clock}_C: & P(*) \rightarrow Clock_C(t) \\
\text{clock}_K: & P(*) \rightarrow Clock_K(t)
\end{array}$$

There should be additional 4 facts for role state predicates and another fact for the network predicate. Additional facts representing memory, clock and validity constraints, *i.e.*$Auth$, $Service$, $DoneMut_C$, Mem_S, $Clock_C$, $Clock_K$, $Valid_K$, $Valid_T$, $Valid_S$, require 3 facts (not all are persistent so we don't need all 8 facts).

The trace representing the protocol run with no intruder present is shown below:

$$
\begin{array}{l}
W\ C_0(C)\ K_0(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*)P(*) \rightarrow_{clock_C} \\
W\ C_0(C)\ K_0(K)\ T_0(T)\ S_0(S)\ Clock_C(t_C)\ P(*)P(*)P(*) \rightarrow_{C1} \\
W\ C_1(C, T, n_1, n_2, t_C)\ \ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T) \\
\quad N(\langle Cert_C, \langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle\rangle)\ P(*)P(*)P(*) \rightarrow_{constraint_K} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ Valid_K(C, T, n_1) \\
\quad N(\langle Cert_C, \langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle\rangle)\ P(*)P(*) \rightarrow_{clock_K} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ Valid_K(C, T, n_1)\ Clock_K(t_K) \\
\quad N(\langle Cert_C, \langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle\rangle)\ P(*) \rightarrow_{K1} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*) \\
\quad N(\langle enc(pk_C, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle), \\
\qquad\qquad \langle C, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{C2} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad N(\langle C, \langle\langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{constraint_T} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Valid_T(C, S, n_2) \\
\qquad N(\langle C, \langle\langle enc(k_T, \langle AKey, C\rangle), enc(k_C, \langle AKey, \langle n_1, T\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{T1} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ P(*)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad N(\langle C, \langle enc(k_S, \langle SKey, C\rangle), enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{clock} \\
W\ C_2(C, T, S, AKey, n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ Clock_C(t)P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad N(\langle C, \langle enc(k_S, \langle SKey, C\rangle), enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{C3} \\
W\ C_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(K)\ T_1(T)\ S_0(S)\ P(*) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad Service(enc(k_S, \langle SKey, C\rangle, S, SKey) \\
\qquad N(\langle enc(k_S, \langle SKey, C\rangle, enc(SKey, \langle C, t\rangle)\rangle\rangle) \rightarrow_{constraint_S} \\
W\ C_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(K)\ T_1(T)\ S_0(S) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
\qquad Service(enc(k_S, \langle SKey, C\rangle, S, SKey)\ Valid_S(C, t) \\
\qquad N(\langle enc(k_S, \langle SKey, C\rangle, enc(SKey, \langle C, t\rangle)\rangle\rangle) \rightarrow_{S1} \\
W\ C_3(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(K)\ T_1(T)\ S_1(S) \\
\qquad Service(enc(k_S, \langle SKey, C\rangle, S, SKey)\ Mem_S(C, SKey, t) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ N(enc(SKey, t)) \rightarrow_{C4} \\
W\ C_4(C, S, SKey, t, enc(k_S, \langle SKey, C\rangle))\ K_1(K)\ T_1(T)\ S_1(S) \\
\qquad Service(enc(k_S, \langle SKey, C\rangle, S, SKey)\ Mem_S(C, SKey, t) \\
\qquad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ DoneMut_C(S, SKey)
\end{array}
$$

Appendix H.1. Man-In-The-Middle attack on PKINIT

A Man-in-the-middle attack on PKINIT is informally shown in Figure H.32. For this attack to succeed intruder has to be a legitimate Kerberos client so that the KAS server could grant him credentials. We model that by introducing a compromised client B whose keys and certificates are known to intruder.

$$
\begin{array}{l}
C \longrightarrow I(K) : Cert_C, \{t_C, n_2\}_{sk_C}, C, T, n_1 \\
I(C) \longrightarrow K : Cert_B, \{t_C, n_2\}_{sk_B}, B, T, n_1 \\
K \longrightarrow I(C) : \{Cert_K, \{k, n_2\}_{sk_K}\}_{pk_B}, B, \{AKey, C\}_{k_T}, \{AKey, n_1, t_K, T\}_k \\
I(K) \longrightarrow C : \{Cert_K, \{k, n_2\}_{sk_K}\}_{pk_C}, C, \{AKey, C\}_{k_T}, \{AKey, n_1, t_K, T\}_k \\
C \longrightarrow G : \{AKey, C\}_{k_T}, \{C\}_{AKey}, C, S, n_3 \\
G \longrightarrow C : C, \{SKey, C\}_{k_S}, \{SKey, n_3, S\}_{AKey} \\
C \longrightarrow S : \{SKey, C\}_{k_S}, \{C, t_{c,S_{req}}\}_{SKey} \\
S \longrightarrow C : \{t_{c,S_{req}}\}_{SKey}
\end{array}
$$

Figure H.32: Man-in-the-middle attack on PKINIT Protocol.

This flaw allows an attacker to impersonate Kerberos administrative principals and end-servers to a client, hence breaching the authentication guarantees of Kerberos PKINIT. It also gives the attacker the keys that the server K would normally generate to encrypt the service requests of this client, hence defeating confidentiality as well. The consequences of this attack are quite serious. For example, the attacker could monitor communication between an honest client and a Kerberized network file server. This would allow the attacker to read the files that the client believes are being securely transferred to the file server.

Initial set of facts has 17 facts:

$$
\begin{array}{rl}
W = & Client(C, pk_C)\ KP(pk_C, sk_C)\ AnnK(pk_C) \\
& Client(B, pk_B)\ KP(pk_B, sk_B)\ AnnK(pk_B) \\
& M_{ek}(pk_B)\ M_{dk}(sk_B)\ M_g(B)\ M_p(Cert_B) \\
& KAS(K)\ KP(pk_K, sk_K)\ AnnK(pk_K) \\
& TGS(T)\ TGSKey(T, k_T)\ \ Server(S)\ ServerKey(S, k_S)\ .
\end{array}
$$

There should be additional 4 facts for role state predicates and another fact for the network predicate. Memory, clock and validity constraints, *i.e.*$Auth$, $Service$, $DoneMut_C$, Mem_S, $Clock_C$, $Clock_K$, $Valid_K$, $Valid_T$, $Valid_S$, require 3 additional facts.

The attack requires a configuration of at least **31 facts** (21 for the protocol and additional 10 for the intruder) of the **size 28**, as shown by the following trace.

$$
\begin{aligned}
&W\ C_0(C)\ K_0(K)\ T_0(T)\ S_0(S)\ R(*)R(*)R(*)R(*)R(*)R(*)\\
&\quad P(*)P(*)P(*)P(*)R(*)R(*)R(*)R(*)R(*)R(*) \rightarrow_{(clock_C, C1)}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ \ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)R(*)\\
&\quad R(*)R(*)R(*)R(*)R(*)\ N_S(\langle Cert_C, \langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle\rangle) \rightarrow
\end{aligned}
$$

Intruder has to intercept and digest the message in order to modify it.

$$
\begin{aligned}
&\rightarrow_{REC}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ \ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\\
&\quad D(\langle Cert_C, \langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle\rangle)\\
&\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*)P(*) \rightarrow_{DCMP}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ \ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)P(*)\\
&\quad D(Cert_C)\ D(\langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle)\ R(*)R(*)R(*)R(*) \rightarrow_{DELD}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ \ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)P(*)\\
&\quad B(*)\ D(\langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle)\ R(*)R(*)R(*)R(*) \rightarrow_{DCMPB}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ \ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)P(*)\\
&\quad D(enc(sk_C, \langle t_C, n_2\rangle))\ D(\langle C, \langle T, n_1\rangle\rangle)\ R(*)R(*)R(*)R(*) \rightarrow_{DSIG}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ \ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)P(*)\\
&\quad D(\langle t_C, n_2\rangle)\ M_c(enc(sk_C, \langle t_C, n_2\rangle))\ D(\langle C, \langle T, n_1\rangle\rangle)\ R(*)R(*)R(*) \rightarrow_{DELMB}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)P(*)\\
&\quad D(\langle t_C, n_2\rangle)\ B(*)\ D(\langle C, \langle T, n_1\rangle\rangle)\ R(*)R(*)R(*) \rightarrow_{DM}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)P(*)\\
&\quad M_s(\langle t_C, n_2\rangle)\ B(*)\ D(\langle C, \langle T, n_1\rangle\rangle)\ R(*)R(*)R(*) \rightarrow_{DCMPB}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\\
&\quad M_s(\langle t_C, n_2\rangle)\ D(C)\ D(\langle T, n_1\rangle)\ R(*)R(*)R(*)P(*)P(*)P(*)P(*) \rightarrow_{DM}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\\
&\quad M_s(\langle t_C, n_2\rangle)\ D(C)\ M_s(\langle T, n_1\rangle)\ R(*)R(*)R(*)P(*)P(*)P(*)P(*) \rightarrow_{(LRNG)}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\\
&\quad M_s(\langle t_C, n_2\rangle)\ M_g(C)\ M_s(\langle T, n_1\rangle)\ R(*)R(*)P(*)P(*)P(*)P(*) \rightarrow
\end{aligned}
$$

Intruder starts composing the modified message replacing $Cert_C$, C and C's signature with $Cert_B$, B and B's signature. Since B is compromised intruder knows all the required data.

$$
\begin{aligned}
&\rightarrow_{(USES, USEG, COMP, USES, SIG)}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)P(*)\\
&\qquad M_s(\langle t_C, n_2\rangle)\ M_g(C)\ M_s(\langle T, n_1\rangle)\\
&\qquad C(\langle I, \langle T, n_1\rangle\rangle)\ C(enc(sk_B, \langle t_C, n_2\rangle)) \rightarrow
\end{aligned}
$$

At this point intruder has no $R(*)$ facts left.

$$
\begin{aligned}
&\rightarrow_{(COMP,USEP,COMP)}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\\
&\qquad M_t(t_C)\ M_n(n_2)\ M_g(C)\ M_g(T)\ M_n(n_1)\ P(*)P(*)P(*)P(*)R(*)\\
&\qquad C(\langle Cert_B, \langle enc(sk_B, \langle t_C, n_2\rangle), \langle B, \langle T, n_1\rangle\rangle\rangle\rangle) \rightarrow_{SND}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\ P(*)P(*)P(*)R(*)R(*)\\
&\qquad M_t(t_C)\ M_n(n_2)\ M_g(C)\ M_g(T)\ M_n(n_1)\\
&\qquad N_R(\langle Cert_B, \langle enc(sk_B, \langle t_C, n_2\rangle), \langle B, \langle T, n_1\rangle\rangle\rangle\rangle) \rightarrow_{DEL^4}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ TGS(T)\\
&\qquad M_g(C)\ P(*)P(*)P(*)R(*)R(*)R(*)R(*)R(*)R(*)\\
&\qquad N_R(\langle Cert_B, \langle enc(sk_B, \langle t_C, n_2\rangle), \langle B, \langle T, n_1\rangle\rangle\rangle\rangle) \rightarrow
\end{aligned}
$$

Intruder sends the modified message to K and deletes some of the data from the memory, keeping the name of the client in the memory for later use.

$$
\begin{aligned}
&\rightarrow_{constraint_K}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ M_g(C)\ Valid_K(C,T,n_1)\\
&\qquad N_R(\langle Cert_B, \langle enc(sk_B, \langle t_C, n_2\rangle), \langle B, \langle T, n_1\rangle\rangle\rangle\rangle)\\
&\qquad R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*) \rightarrow_{clock_K}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_0(K)\ T_0(T)\ S_0(S)\ M_g(C)\ Valid_K(C,T,n_1)\ Clock_K(t_K)\\
&\qquad N_R(\langle Cert_B, \langle enc(sk_B, \langle t_C, n_2\rangle), \langle B, \langle T, n_1\rangle\rangle\rangle\rangle)\\
&\qquad R(*)R(*)R(*)R(*)R(*)R(*)P(*) \rightarrow_{K1}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\\
&\qquad R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*)\\
&\qquad N_S(\langle enc(pk_B, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle),\\
&\qquad\qquad\qquad \langle B, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle\rangle) \rightarrow
\end{aligned}
$$

Intruder intercepts the message intended for C and decomposes it cleverly, *i.e.*uses the already existing submessages and only decomposes what's necessary for learning the information contained.

$$
\begin{aligned}
&\rightarrow_{REC}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\\
&\qquad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*)P(*)\\
&\qquad D(\langle enc(pk_B, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle),\\
&\qquad\qquad \langle B, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{DCMP}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ P(*)P(*)P(*)P(*)\\
&\qquad D(enc(pk_B, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle))\ R(*)R(*)R(*)R(*)\\
&\qquad D(\langle B, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle) \rightarrow
\end{aligned}
$$

$$
\begin{array}{l}
\rightarrow_{DEC} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C) \\
\qquad R(*)R(*)R(*)P(*)P(*)P(*)P(*) \\
\qquad M_c(enc(pk_B, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle))\ D(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle) \\
\qquad D(\langle B, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle) \rightarrow_{DELMC} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C) \\
\qquad R(*)R(*)R(*)P(*)P(*)P(*)P(*)B(*)\ D(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle) \\
\qquad D(\langle B, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle) \rightarrow_{DCMPB} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ P(*)P(*)P(*)P(*) \\
\qquad B(*)\ M_s(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle)\ R(*)R(*)R(*) \\
\qquad D(\langle B, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle) \rightarrow_{DM} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ P(*)P(*)P(*)P(*) \\
\qquad M_s(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle)\ R(*)R(*)R(*) \\
\qquad D(B)D(\langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle) \rightarrow_{DELD} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ P(*)P(*)P(*)P(*) \\
\qquad M_s(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle)\ R(*)R(*)R(*) \\
\qquad B(*)D(\langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle) \rightarrow_{DM} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ P(*)P(*)P(*)P(*) \\
\qquad M_s(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle)\ R(*)R(*)R(*) \\
\qquad B(*)M_s(\langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle) \rightarrow
\end{array}
$$

Intruder starts composing the message form the parts of the intercepted message and the data stored previously.

$$
\begin{array}{l}
\rightarrow_{USES} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ P(*)P(*)P(*)P(*) \\
\qquad M_s(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle)\ R(*)R(*) \\
\qquad B(*)M_s(\langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle) \\
\qquad C(\langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle) \rightarrow_{USEG} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ R(*)P(*)P(*)P(*)P(*) \\
\qquad M_s(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle) \\
\qquad B(*)M_s(\langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle) \\
\qquad C(C)\ C(\langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle) \rightarrow_{COMP} \\
W\ C_1(C, T, n_1, n_2, t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ P(*)P(*)P(*)P(*) \\
\qquad M_s(\langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle)\ R(*)R(*) \\
\qquad B(*)M_s(\langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle) \\
\qquad C(\langle C, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle) \rightarrow
\end{array}
$$

$$
\begin{array}{l}
\rightarrow_{USES} \\
W\ C_1(C,T,n_1,n_2,t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ R(*)P(*)P(*)P(*)P(*) \\
\quad M_s(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)\ C(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle) \\
\quad B(*)M_s(\langle enc(k_T,\langle AKey,C\rangle), enc(k,\langle AKey,\langle n_1,\langle t_K,T\rangle\rangle\rangle)\rangle) \\
\quad C(\langle C,\langle enc(k_T,\langle AKey,C\rangle), enc(k,\langle AKey,\langle n_1,\langle t_K,T\rangle\rangle\rangle)\rangle\rangle) \rightarrow_{SIG} \\
W\ C_1(C,T,n_1,n_2,t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ R(*)P(*)P(*)P(*)P(*) \\
\quad M_s(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)\ C(enc(pk_C,\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)) \\
\quad B(*)M_s(\langle enc(k_T,\langle AKey,C\rangle), enc(k,\langle AKey,\langle n_1,\langle t_K,T\rangle\rangle\rangle)\rangle) \\
\quad C(\langle C,\langle enc(k_T,\langle AKey,C\rangle), enc(k,\langle AKey,\langle n_1,\langle t_K,T\rangle\rangle\rangle)\rangle\rangle) \rightarrow_{COMP} \\
W\ C_1(C,T,n_1,n_2,t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\ P(*)P(*)P(*)P(*) \\
\quad M_s(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)\ R(*)R(*) \\
\quad B(*)M_s(\langle enc(k_T,\langle AKey,C\rangle), enc(k,\langle AKey,\langle n_1,\langle t_K,T\rangle\rangle\rangle)\rangle) \\
\quad C(\langle enc(pk_C,\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle), \\
\qquad \langle C,\langle enc(k_T,\langle AKey,C\rangle), enc(k,\langle AKey,\langle n_1,\langle t_K,T\rangle\rangle\rangle)\rangle\rangle\rangle) \\
\rightarrow_{SND,DEL^3}\ W\ C_1(C,T,n_1,n_2,t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C) \\
\qquad R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \\
\qquad N_R(\langle enc(pk_C,\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle), \\
\qquad\quad \langle C,\langle enc(k_T,\langle AKey,C\rangle), enc(k,\langle AKey,\langle n_1,\langle t_K,T\rangle\rangle\rangle)\rangle\rangle\rangle) \rightarrow
\end{array}
$$

In the remaining part of protocol intruder only forwards the messages, *i.e.* plays the role of the network.

$$
\begin{array}{l}
\rightarrow_{C2} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad N_S(\langle C,\langle\langle enc(k_T,\langle AKey,C\rangle), enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{constraint_T} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Valid_T(C,S,n_2)\ P(*) \\
\quad N_S(\langle C,\langle\langle enc(k_T,\langle AKey,C\rangle), enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{FWD} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ R(*)R(*)R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Valid_T(C,S,n_2)\ P(*) \\
\quad N_R(\langle C,\langle\langle enc(k_T,\langle AKey,C\rangle), enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle\rangle) \rightarrow_{T1} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ P(*)P(*) \\
\quad N_S(\langle C,\langle enc(k_S,\langle SKey,C\rangle), enc(AKey,\langle SKey,\langle n_2,S\rangle\rangle)\rangle\rangle) \rightarrow_{clock_C} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Clock_C(t)\ P(*) \\
\quad N_S(\langle C,\langle enc(k_S,\langle SKey,C\rangle), enc(AKey,\langle SKey,\langle n_2,S\rangle\rangle)\rangle\rangle) \rightarrow
\end{array}
$$

$$
\begin{array}{l}
\rightarrow_{FWD} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_1(T)\ S_0(S)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Clock_C(t)\ P(*) \\
\quad N_R(\langle C,\langle enc(k_S,\langle SKey,C\rangle),enc(AKey,\langle SKey,\langle n_2,S\rangle\rangle)\rangle\rangle) \rightarrow_{C3} \\
W\ C_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(K)\ T_1(T)\ S_0(S)\ R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ R(*)R(*)R(*)P(*) \\
\quad Service(enc(k_S,\langle SKey,C\rangle,S,SKey) \\
\quad N_S(\langle enc(k_S,\langle SKey,C\rangle,enc(SKey,\langle C,t\rangle)\rangle\rangle) \rightarrow_{constraint_S} \\
W\ C_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(K)\ T_1(T)\ S_0(S) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad Service(enc(k_S,\langle SKey,C\rangle,S,SKey)\ Valid_S(C,t) \\
\quad N_S(\langle enc(k_S,\langle SKey,C\rangle,enc(SKey,\langle C,t\rangle)\rangle\rangle) \rightarrow_{FWD} \\
W\ C_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(K)\ T_1(T)\ S_0(S) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad Service(enc(k_S,\langle SKey,C\rangle,S,SKey)\ Valid_S(C,t) \\
\quad N_R(\langle enc(k_S,\langle SKey,C\rangle,enc(SKey,\langle C,t\rangle)\rangle\rangle) \rightarrow_{S1} \\
W\ C_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(K)\ T_1(T)\ S_1(S)\ R(*)R(*)R(*) \\
\quad Service(enc(k_S,\langle SKey,C\rangle,S,SKey)\ Mem_S(C,SKey,t)\ R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ N_S(enc(SKey,t)) \rightarrow_{FWD} \\
W\ C_3(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(K)\ T_1(T)\ S_1(S)\ R(*)R(*)R(*) \\
\quad Service(enc(k_S,\langle SKey,C\rangle,S,SKey)\ Mem_S(C,SKey,t)\ R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ N_R(enc(SKey,t)) \rightarrow_{C4} \\
W\ C_4(C,S,SKey,t,enc(k_S,\langle SKey,C\rangle))\ K_1(K)\ T_1(T)\ S_1(S)\ R(*)R(*)R(*) \\
\quad Service(enc(k_S,\langle SKey,C\rangle,S,SKey)\ Mem_S(C,SKey,t)\ R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ DoneMut_C(S,SKey)
\end{array}
$$