SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODJEL

Tajana Ban Kirigin

Složenost u sustavima suradnje koji mogu generirati nove vrijednosti

Disertacija

Voditelji rada: Andre Scedrov i Zvonimir Šikić

Zagreb, 2011.

University of Zagreb

Faculty of Natural Sciences and Mathematics
Department of Mathematics

Tajana Ban Kirigin

Computational complexity of collaborative systems with nonce creation

Dissertation

Supervisors: Andre Scedrov i Zvonimir Šikić

Zagreb, 2011.

In memory of Dean Rosenzweig

Acknowledgments

With sincere gratitude to Professor Andre Scedrov, who guided me through this work with continuous support and encouragement. He has dedicated so much of his precious time and shared many insights and ideas, and that's been a real privilege for me.

I'd also like to thank Vivek Nigam and Professor Max Kanovich for many long and insightful discussions that were essential for the creation of the thesis.

I thank my colleagues and friends at the Department of Mathematics, University of Zagreb and Department of Mathematics, University of Rijeka for their help and support, in particular Professor Šikić, Professor Vuković, Professor Manger and Majda Trobok.

Finally, special thanks to my dearest friends Paola Glavan and Sonja Eberling and my family for always being there for me.

Contents

Introduction

Frameworks based on multiset rewrite systems have been recently proposed for modeling *collaborative systems* [26, 27]. In such systems, participants, or agents collaborate in order to reach a state (or system configuration) which contains some common goal. They perform actions, specified by rewrite rules, which change the system's state of the world, specified by a multiset of facts. A sequence of actions that leads from an initial configuration to a configuration containing the common goal is often called a *plan*.

Since information can be passed from one agent to another while agents do not necessarily trust each other completely, one is interested in showing that the system is secure, i.e. some critical states cannot be reached. Such critical states do not appear only in the domain of computer security, but in information security in general: a configuration where an agent's sensitive information, such as a password, is leaked to other agents is an example of such critical state that should be avoided. In [26, 27], this issue is addressed by different notions of policy compliance for a system. The first, called *system compliance*, is satisfied if there is no way for the agents to reach a critical configuration. The second, called *weak plan compliance*, is satisfied if there is a plan for which no critical configuration is reached. Finally, the third notion, called *plan compliance*, is satisfied by the system if no critical configuration for one agent can be reached if this agent decides to no longer collaborate, that is, when only the other agents perform actions.

In the light of the real-life restrictions of resources, agents in such systems often have a limited storage capacity, that is, at any moment an agent can remember at most a bounded number of facts. In order to reflect this intuition, these frameworks use actions with same number of facts in their pre- and post-conditions, called *balanced* actions. With such actions all configurations in a run have the same size as the initial configuration.

We build on the framework in [26, 27]. On the one hand, we make two extensions to the framework: We allow actions to create fresh values, which adds to the expressivity of the systems. Fresh values, often called nonces in protocol security literature, are essential not only in protocol security but also in other administrative processes that require for example unique identification: A fresh value is assigned to a bank transaction, so that transactions are uniquely identified. Next, we introduce among the available agents a *leader* who is trusted by all other agents. Each agent can also interact with the leader directly.

On the other hand, we impose two restrictions to the framework: We allow balanced actions to change exactly one fact, a, to another fact, a', and in the process, one is allowed to check the presence of a fact, b, which can also be seen as checking for a condition. Furthermore, we introduce a class of collaborative systems called *progressing collaborative systems* by not allowing the same instance of a balanced action to be used more than once. This restriction incorporates the assumption that systems are progressing: each transition rule can be seen as checking one of the check-boxes in a to-do list necessary to achieve a final goal. Whenever one action is performed, one never needs to repeat this action again. A similar notion of progressing also appears in [18], where it was introduced to protocol theories through protocol role states.

Under both of the restrictions above we show that in our model the weak compliance problem is NP-hard. It follows from [2, 36] that it is also in NP. Moreover, we also observe that if we allow balanced actions to be used more than once, the same problem for the resulting system becomes PSPACE-hard. The PSPACE upper bound follows from [27]. The same problem is also PSPACE-hard if we allow balanced actions to create values with fresh ones, but if each instance of a balanced action is used at most once. The upper bound for this case with balanced actions is open and left for future work. We prove our PSPACE-hard lower bounds by encoding Turing machines in our framework. The main challenge here is to *faithfully* simulate the behavior of a Turing machine that uses a sequential, non-commutative tape in our formalism that uses commutative multisets. This contrasts with the encoding of Turing machines in [22, p. 469] where the tape is encoded using non-commutative matrices.

Since rewrite rules can be used to model protocols and the relevant security problems, see [18], as an application we turn to security protocol analysis. In contrast with traditional intruder models, such as in protocol security, which normally include a powerful Dolev-Yao intruder [17] that has an unbounded memory, balanced systems imply that all players inside our system, including inside adversaries, have a bounded storage capacity, that is, they can only remember at any moment a bounded number of facts. On the other hand, our adversaries and the standard Dolev-Yao intruder [17] share many capabilities, namely, they are able, within their bounded storage capacity, to compose, decompose, overhear, and intercept messages as well as create fresh values.

We show that the secrecy problem of whether or not an adversary can discover a secret is PSPACE-complete when actions are balanced and can create fresh values. This contrasts with previous results in protocol security literature [18], where it is shown that the same problem is undecidable. However, there the intruder was allowed to have unbalanced actions, or in other words, it was assumed that the intruder's memory was not necessarily bounded. We further investigate the consequences of our results in the domain of protocol security. In particular, we demonstrate that when our adversary has *enough* storage capacity, then protocol anomalies, such as the Lowe anomaly [30] of the Needham-Schroeder public key exchange protocol, can also occur.

This dissertation is structured as follows: in Section 1 we review the main definitions of local state transition systems used to model collaborative systems with policy compliance. We introduce progressing systems and extend the systems with nonce generation. We then show connections to linear logic with focusing. In Section 2 we point to existing problems and show how to circumvent them by formalizing a specific notion of freshness for balanced systems. In Section 3 we summarize the main theoretical results involving complexity of the different problems considered. We introduce an intruder model with bounded memory in Section 4 and demonstrate some anomalies in Section 4.4 and 5. Finally, in Section 6, we review related work and we conclude by pointing out to future work.

Chapter 1

Collaborative Systems

Collaboration among organizations or individuals is common. It's main purpose is to achieve some common goal. It necessarily involves some information flow. While much of the information shared may be general and harmless, some of the information is sensitive, such as detailed sales reports at an organizational level or social security numbers and credit card numbers at an individual level. The need to share information and the desire to keep it confidential are two competing notions which affect the outcome of a collaboration.

Models formalizing collaborative systems have been introduced in [26, 27]. In this section we review the main vocabulary and concepts introduced in [26, 27] and also extend the definitions to accommodate a leader and actions that can create fresh values. In addition to the local state transition systems introduced [26, 27] we will have two special views on the agents of the system. The first one distinguishes among the agents a special agent called the leader and the second one models the agent which is adversary inside the system. We will point out the particular LSTS as we introduce the problems we study.

The model has four main components. Being an evolving system, there must be a way of describing the configuration of the system at any given moment. Additionally, we describe how the agents transform the configurations from one to another via local actions. In our setting the agents also have some (common) goal of the collaboration, as well as (distinct) confidentiality concerns. These concerns are expressed as confidentiality policies.

We introduce each of these four components in detail in the next subsections. We start by giving examples of the scenarios we model by our collaborative systems.

Medical Test Scenario. Consider the scenario 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 secretary, a nurse, and a lab technician. Each of these agents have their own set of tasks. For instance, the patient's initial task could be to make an appointment and go to the hospital. Then, the secretary would send the patient to the nurse who would collect the patient's blood sample and send it to the lab technician, who would finally perform the required test. In such process of getting doctor's professional opinion and eventually receiving a treatment or a prescription, a patient must be willing to share some personal information. Although he might give some information to the nurse and even more personal information to the doctor, he might not be willing to do the same with the hospital receptionist or other patients in the hospital's waiting room. Some of the hospital policies are imposed by law, others are specific for a particular institution. The patient will necessarily feel more comfortable coming into a hospital with policies that would guarantee a high level of confidentiality.

Scientific Research Scenario. As another example, consider a group of scientists working on a joint paper. During such scientific research, scientists must find a balance between sharing too much information and not enough. On one hand, a principal goal of research is to obtain and publish results. On the other hand, if researchers provide their raw data to others too soon, another group may announce the results first. Scientific data sharing also has the converse problem. Scientists often rely on data from outside sources.

1.1 Local State Transition Systems

At the lowest level, we have a first-order signature Σ that consists of a set of sorts together with the predicate symbols $P_1, P_2, \ldots$, function symbols $f_1, f_2, \ldots$, and constant symbols $c_1, c_2, \ldots$ all with specific sorts (or types). The multi-sorted terms over the signature are expressions formed by applying functions to arguments of the correct sort. Since terms may contain variables, all variables must have associated sorts.

Definition 1.1. 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 1.2. A *state*, or a *configuration* of the system is a finite multiset W of facts.

We use both WX and W, X to denote the multiset resulting from the multiset union of W and X. For any fact $P(\vec{t})$ we use $P(\vec{t})^k$ to indicate that there are k instances of the fact $P(\vec{t})$ in the configuration.

As in [26, 27], we assume that the global configuration is partitioned into different local configurations each of which is accessible only to one agent. There is also a public configuration which is accessible to all agents involved in the collaboration. We are considering interactions that take place in a closed-room setting, so we can ignore concerns about an outside intruder. However, differently from [26, 27], when we will be modeling protocols, we assume that among the agents in the system, there is an agent M that behaves like an adversary. We also assume the existence of a special constant s in Σ denoting the secret that should not be discovered by the adversary.
The separation of the global configuration is done by partitioning the set of predicate symbols in the signature. We typically annotate a predicate symbol with the name of the agent that owns it or with *pub* if it is public. For instance, the fact $F_A(\vec{t})$ belongs to the agent A, while the fact $F_{pub}(\vec{t})$ is public. The predicate symbols act like fields of a database in which information may be stored, and the syntactic form of the predicate determines if the fact is local to a given agent or if it is public. The global configuration is the multiset union of the local configurations and the public configuration:

$$Z = Z_{A_1}, Z_{A_2}, \ldots, Z_{A_n}, Z_{pub}$$

where Z_{A_i} is the local configuration of agent A_i and Z_{pub} is the public configuration. More generally, we use X_A to denote any multiset of facts all of which are local to agent A, and X_{pub} to denote any multiset of facts all of which are publicly accessible.
As in [26, 27], each agent has a finite set of rules or actions which transform the global configuration. Under the interpretation of the local predicates as accessibility restrictions it is natural to restrict the actions to conform to that interpretation. Thus the actions are local in the sense that each action can only depend on the local facts of at most one agent:

$$X_A X_{pub} \ \rightarrow_A \ Y_A Y_{pub} \ .$$

The agent who owns an action is specified by the subscript on the arrow. For simplicity, we often omit the name of the agent from the action and from predicates when the agent is clear from the context.

Intuitively, agent A does not have neither read nor write access to any of the predicates which are local to other agents. As actions are multiset rewriting rules, the pre-conditions must be present in the configuration for the action to be enabled. By applying the action, the pre-conditions are erased and replaced with the post-conditions. Since several actions may be enabled simultaneously, we assume the actions are applied nondeterministically.

There is a small detail that deserves some attention. A public fact is accessible by any agent, but only in principle. An agent might not have any action with that fact in the pre-conditions, in which case the term is actually inaccessible to that agent. This is in stark contrast to the standard Dolev-Yao intruder [17] who is always assumed to be able to read messages from the network. No such assumption is made about the agents in our system.

Although some or all of the predicates in a first-order action will contain free variables, we will often abuse terminology and call these predicates facts. So for example, when we count the number of facts in the pre-condition of a first-order action, we are really counting the number of facts that occur under any substitution. Since this number does not change under different substitutions, there is no ambiguity. Given a set of first-order actions, it is possible to propositionalize them by applying every possible substitution. If the underlying signature Σ is finite, then this process will expand the size of the set of actions exponentially. We will be explicit about whether an action is first-order or propositional whenever the distinction is important. In all other cases we will simply use the word action.

1.1.1 Systems that Can Create Fresh Values

Here, as in [9, 18], we allow agents to have more general actions that can create *fresh values*. These values are often called *nonces* in protocol security literature. Such 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 values, 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. Actions that belong to an agent A and create fresh values have the form:

$$X_A X_{pub} \;\rightarrow_A\; \exists \vec{n}. Y_A Y_{pub} \,.$$

The multisets X_A and Y_A contain facts belonging to the agent A and the multisets X_{pub} and Y_{pub} contain only public facts. Actions work as multiset rewrite rules. All free variables in a rule are treated as universally quantified. Facts $X_A X_{pub}$ are the pre-condition of the action and facts $Y_A Y_{pub}$ are the post-condition of the action. By applying the action for a ground substitution (σ), the pre-condition applied to this substitution ($X_A\sigma X_{pub}\sigma$) is replaced with the post-condition applied to the same substitution ($Y_A\sigma Y_{pub}\sigma$). In this process, the existentially quantified variables ($\vec{n}$) appearing in the post-condition are replaced by fresh variables. The rest of the configuration remains untouched. Thus, for example, we can apply the action $P_A(x)\ Q_{pub}(y) \to_A \exists z.R_A(x,z)\ Q_{pub}(y)$ to the global configuration $V\ P_A(t)\ Q_{pub}(s)$ to get the global configuration $V\ R_A(t,c)\ Q_{pub}(s)$, where the constant c is fresh.

Medical Test Scenario. The following multiset rewriting rules specify some of the actions of the agents N (nurse) and L (lab technician) from the medical test scenario:

$$\begin{aligned}&\mathit{Nurse}_N(\text{blank},\text{blank},\text{blank})\ \mathit{Patient}(\mathit{name},\mathit{test})\\ &\qquad\to_N\ \mathit{Nurse}_N(\mathit{name},\text{blank},\mathit{test})\ \mathit{Patient}(\mathit{name},\mathit{test})\end{aligned}$$

$$\begin{aligned}\mathit{Nurse}_N(x,\text{blank},\text{blood}) &\to_N \exists \mathit{id}.\mathit{Nurse}_N(x,\mathit{id},\text{blood})\\ \mathit{Nurse}_N(x,\mathit{id},\text{blood}) &\to_N \mathit{Lab}(\mathit{id},\text{blood})\ \mathit{Nurse}_N(x,\mathit{id},\text{blood})\\ \mathit{Lab}(\mathit{id},\text{blood}) &\to_L \mathit{TestResult}(\mathit{id},\text{result})\end{aligned}$$

Predicates *Patient*, *Lab* and *TestResult* are public, while the predicate Nurse_N is private 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 outcome, while *test*, *name*, x and *id* are variables. The most interesting action is the second action which generates a fresh value, an identification number, that is used for anonymising the patient. Each patient (sample) should have a different identification number assigned. In that way the nurse is able to anonymise the blood samples before passing them on to the lab, so that the lab technicians receive samples with no name of the patient attached, only identifiaction numbers. This is important for privacy concerns, as will be discussed in more detail later.

In addition to the communication through public facts, we allow an agent to safely communicate through a private channel with a privileged agent called *leader*. This is different to communication introduced in [26, 27]. It is accomplished by further partitioning the predicates in the signature. We annotate predicate symbols belonging to a private channel between an agent A and the leader l with Al. Whenever an agent and the leader need to communicate between each other, *e.g.*, request a new password, they can modify the facts in their corresponding private channel. Therefore we further generalize the rules to allow private communication of an agent with the leader, and have rules of the following form:

$$X_A X_{Al} X_{pub} \to_A \exists \vec{n}.Y_A Y_{Al} Y_{pub} \quad \text{and} \quad X_l X_{Al} X_{pub} \to_l \exists \vec{n}.Y_l Y_{Al} Y_{pub}.$$

An agent that is not the leader owns only actions of the former form, where it also owns the local facts and the facts in the private channel. The leader, on the other hand, owns only rules of the latter form, where it also owns the local facts and any fact in any private channel to any agent.

As an example consider the following rule from the medical test scenario:

$$\mathit{Nurse}_N(x, id, test)\ \mathit{TestResult}(id, y)\ \rightarrow_N\ \mathit{Nurse}_N(x, id, test)\ \mathit{Doctor}_{Nl}(x, y)$$

where the doctor is the leader and the predicate Doctor_{Nl} belongs to the private channel between the nurse and the doctor. The nurse is therefore able to communicate with the doctor through a private channel and safely pass the test results together with the patients name to the doctor.

Having introduced the basic concepts of our systems, namely configurations and agents with their actions, we can now define our local state transition systems which model the collaborative systems.

Definition 1.3. A *local state transition system* T is a triple $\langle \Sigma, I, R_T \rangle$, where Σ is the signature underlying the language, $I = \{A_1, \ldots, A_n\}$ is the set of agents, and R_T is the set of (local) actions owned by those agents.

We often use the abbreviation LSTS for a local state transition system, as well as abbreviating it to the one word system. When we consider an LSTS with a leader, we have a tuple $\langle \Sigma, I, l, R_T \rangle$ where $l \in I$ is the leader and when we consider an LSTS with the inside adversary we have a tuple $\langle \Sigma, I, M, R_T, s \rangle$ where $M \in I$ is the intruder and $s \in \Sigma$ the constant representing the secret. Later, we will impose restrictions to the types of actions allowed in a system as discussed in the Introduction.

Since we are considering situations in which the agents are collaborating, we need a way of expressing the goals of collaboration as well as the reachability of those goals. Naturally, the reachability of a configuration depends on what actions are available to the agents. That is, systems with different actions behave differently, so we need a way to specify which system we are studying.
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. For example, in the medical test scenario, a patient is only interested to know if his test results are ready and is not interested in the results of other agents. 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. We may drop the subscript T for the system and A for the agent if the system or the agent is clear from the context.

Definition 1.4. Given an initial configuration W and a partial configuration Z, a *plan* is any sequence of actions that leads from state W to a state containing Z, i.e. $W \rightsquigarrow^* Z$. We say that a plan exactly leads from W to Z when $W \rhd Z$.

Along a plan $r_1, \ldots, r_n$ leading from W to Z we reach configurations X_i so that $X_1 = W$, $Z \subset X_n$ and $X_i \rhd_T^1 X_{i+1}$ for $1 \leq i < n$. We also say that actions r_i and configurations X_i are contained in the plan.

Since we're interested in formalizing systems of constant size, that is systems where all agents have bounded memory, we first need to define how we measure the size of the memory in the system.

Definition 1.5. The *size of a fact* is the number of term and predicate symbols it contains. We count one for each predicate and function name, and one for each variable or constant symbol. We use $|F|$ to denote the size of a fact F.

For example, $|F(x, c)| = 3$, and $|F(f(x, n), z)| = 5$.

Definition 1.6. The size of a configuration $\mathcal{S}$ is the number of public and private facts in $\mathcal{S}$.

Generally, actions in LSTS can increase or decrease the number of facts in the global configuration. When modeling systems with a fixed size, we should consider only rules that do not change the number of facts of the global configuration, *i.e.* the size of the configuration, and furthermore, it is also important to impose a bound on the size of facts.

Definition 1.7. A *balanced action* is an action that has the same number of facts in the pre-condition as the post-condition, counting multiplicity. If an action is not balanced then it is *unbalanced.*
A system is called balanced if every action of the system is balanced. Otherwise, the system is called unbalanced.

For example the action:

$$Nurse_N(x, \text{blank}, \text{blood}) \rightarrow_N \; \exists id. Nurse_N(x, id, \text{blood})$$

from the medical test scenario is balanced while the following action is unbalanced:

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

A total size of the configuration of a balanced system is constant. There is a fixed number of facts in a 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. Intuitively, this restriction forces each agent to have a buffer or database of a fixed size before the collaboration. The agents may update values in this buffer and erase values to leave empty slots, but they may not change the size of the buffer. Notice that if we use balanced systems but do not impose an upper bound to the size of facts, then we do not necessarily impose a bound on the memory of agents. This is because agents could use functions and facts with unbounded depth to remember as many constants (or data) they need. For instance, 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 a single fact $Q(\langle c_1, \langle c_2, \langle \cdots, \langle c_{n-1}, c_n \rangle\rangle \cdots \rangle\rangle)$ and the pairing function $\langle \cdot, \cdot \rangle$. Intuitively, by using balanced systems and assuming an upper bound on the size of facts, we obtain a bound on the number of memory slots available in any configuration in a plan or in a run of a protocol. Notice as well that such upper bound on the size of facts was also assumed in previous work [18, 26, 27].

Later we will specifically look at systems with actions restricted to change exactly one fact in the configuration. We call such actions *monadic actions.*

Definition 1.8. A rule is *monadic* if it changes exactly one state variable, possibly checking a single condition, i.e. it has one of the following forms:

$$a \to a', \qquad ab \to a'b \qquad \text{and} \qquad ab \to \exists \vec{n}. a'b.$$

The first type of mondic rules is also called *context-free*, and the second and the third type are also called *context-sensitive.*

A context-free rule changes one state variable, a, to another state variable, a', without checking for the presence of any other variable. A context-sensitive rule check for the presence of a state variable b, which can be seen as a condition for applying such a rule. Moreover, in the third type of the rule new constants, $\vec{n}$, are created, while in the first and second type of rules no new constants are created.

1.1.2 Policy Compliances

In order to achieve a final goal, it is often necessary for an agent to share some private knowledge with some other agent. For example, in the medical scenario, the patient needs to share his name with the secretary in order for the test to be run. 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 client could share his password with the bank, which is trusted, and not with another client. One is, therefore, interested in determining whether a system complies with some *confidentiality policies*, *e.g.* the secretary should not know the test results of the patient.

Such confidentiality concerns appear in the domain of computer security and in information security in general: a configuration where an agent's sensitive information, such as a password, is leaked to other agents is an example of such *critical state* that should be avoided.

In formalizing what it means for an agent's secrets to leak, we need to know where that agent is willing to let his information flow. We will assume that each agent has a data confidentiality policy which specifies which pieces of data other agents are prohibited from learning.

Definition 1.9. The *confidentiality policy* of an agent is a set of partial configurations that the agent views as undesirable or bad. A configuration is called *critica*l for A if it contains one of the partial configurations from A's policy. A configuration is simply called critical if it is critical for some agent.

Notice that since policies only specify which configurations must not occur, no conflict arises when combining policies. The combination of two policies simply specifies the union of their critical configurations.

When modeling real situations these policies serve as constraints in the model, which may not represent real constraints in the actual system. For example a policy which is mandated by law will impose real restrictions on the system, while a patients policy may simply represent his personal preferences.

The agents' confidentiality policies are simply sets of configurations which the agents want to avoid. However, they may want to avoid them in a looser or stricter sense depending on the level of trust the agents have amongst themselves. We review three types of policy compliances proposed in [26, 27].

Definition 1.10. (System compliance) A local state transition system T in initial configuration W, given a set of confidentiality policies, is called *compliant* if no critical state is reachable from W.

Compliant systems may be viewed as well-designed systems. Agents have the guarantee that the actions of the system do not allow their critical configurations to be reached, whether by the malicious collusion of other agents or by the careless use of their own actions. A compliant system dispenses with all confidentiality concerns before the collaboration even starts, and the agents can then focus on the separate problem of finding a plan which leads to a goal configuration. Notice, however, that system compliance depends on the initial configuration of the system. It is possible for a system to be compliant in one initial configuration, and not be compliant in another. System compliance protects each agent not only from any possible sequence of actions other agents can take, it also protects each agent from their own actions.

This interpretation is most appropriate when the agents share a very low level of trust. For example, a company may ask employees or external collaborators to sign a non-disclosure agreement. System compliance is also appropriate for the medical test example. No matter what happens, patients' sensitive data should not be compromised, regardless what actions the agents involved take. According to hospital policies, it should never be possible that, for example, a patient's test results publicly leak together with the patient's name.

System compliance is a very strong notion. However, for many scenarios this interpretation of policies might be too strong. For this reason it is useful to consider weaker definitions that can still provide some level of protection.

Definition 1.11. (Weak plan compliance) Given a local state transition system T, an initial configuration W, a (partial) goal configuration Z, and a set of confidentiality policies, a plan is said to be *weakly compliant* if none of the configurations contained in the plan are critical for any agent.

While system compliance requires that for any sequence of actions no critical state is reachable from the initial configuration, weak plan compliance only requires the existence of a plan that does not reach a critical state. Weak plan compliance may be appropriate for a set of agents who are mutually trusting. If a system has a compliant plan leading to a goal then each agent knows two things. Firstly, they know that there is plan leading from the initial configuration to the goal configuration. Secondly, they know that none of their critical configurations will be reached as long as everybody follows the plan. This means that the agents must trust each other to follow the plan because if they deviate from the plan, then all guarantees of compliance are lost.

As an example consider what happens when Alice eats at a restaurant and wants to pay for the meal with a credit card. In order to do so, she must give her card to the waiter, Bob, who takes it away for validation. She trusts the waiter, Bob, not to write down her card details or share it with anybody else, although she is well aware that he can do so. If she wants to achieve the goal of paying with a credit card, she must necessarily trust the restaurant policy and Bob in particular. If she met Bob in a different social situation, however, she might not give him her credit card in that context. Similarly, she doesnt give her credit card to her friends even though she may trust them. This is because her friends have no legitimate reason to learn her card number.

The third compliance provides an intermediate level of protection between the system compliance and the weak plan compliance. Intuitively, if a plan is compliant it protects those agents who follow the plan against those who may choose to deviate from the plan. Any agent A following the plan is assured that the plan contains no configurations critical for him. In addition, 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. On the other hand, 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.

Definition 1.12. (Plan compliance) Given a local state transition system T, an initial configuration W, a (partial) goal configuration Z, and a set of confidentiality policies, a plan is said to be *compliant* if it is weakly compliant and if 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.

Plan compliance is obviously stronger than weak plan compliance. In contrast to system compliance, this interpretation considers only one plan at a time. Instead of implying that any sequence of actions is safe to use, it implies that a specific sequence is safe to use even if everybody else behaves differently.

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 1.13. 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.

Which compliance is more suitable will depend on the process that is modeled. In some cases, when agents do not trust each other, one might require system compliance.

This may be the case in the medical test scenario: According to hospital policies, it should never be possible that, for example, the test results of the patient are publicly leaked together with patient's name. In other cases, however, when agents are more trusting, one might only require weak plan compliance: A group of researchers proposes their project for a grant and look for a plan for which the proposal is sent to the funding agency before the deadline. Plan compliance might be appropriate for collaboration between competing companies.

In [27] it's been shown that the problem of checking whether a system satisfies such policy compliances is PSPACE-complete under the condition that actions are balanced. For general systems with possibly unbalanced actions, it's been shown in [26] that system compliance is EXPSPACE-complete, while weak plan compliance and plan compliance are undecidable.

This thesis makes the additional assumption that initial and the goal configurations are closed under renaming of nonces.

1.1.3 Progressing Collaborative Systems

Many administrative or business processes have not only a bounded number of transactions, but also have a *progressing behavior*: whenever a transaction is performed, it does not need to be repeated. For instance, whenever one initiates some administrative task, one receives a "to-do" list containing the sub-tasks necessary for accomplishing the final goal. Once one has "checked" an item on the list, one does not need to return to this item anymore. When all items have been checked, the process ends.

For a more concrete example, consider the already introduced medical scenario where a patient needs a medical test to be performed. Such administrative processes are usually progressing: once a patient has made an appointment, he does not need to repeat this action again. Even in cases where it may appear that the process is not progressing, it is. For example, if the patient needs to repeat the test because his sample was spoiled, then a different process is initiated with possibly a new set of actions: the secretary is usually allowed to give the patient priority in scheduling appointments. Moreover, it is not realistic to imagine that one would need to reschedule infinitely many times, but only a very small number of times.

For another example, when submitting a grant proposal, or even this paper, one must submit it before some deadline, otherwise the grant is not accepted. The use of deadlines is another form of bounding processes: since actions take some positive time to be performed, it must be the case that the whole operation is completed within a bounded number of transactions.

Administrative processes not only have a bounded number of transactions, but also manipulate a bounded number of values. Consider, for example, the simple process where a bank customer needs a new PIN number: The bank will either assign the customer a new PIN number, which is often a four digit number and hence bounded. Alternatively, the bank will allow the customer to select a password satisfying some conditions, *e.g.*, all its characters must be alphanumeric and it has to be of some particular length, and hence again bounded. Even when the bank lets the customer select a password of any particular length, in practice this password is bounded since users are never able to use an unbounded password due to buffer sizes, etc.

Definition 1.14. A plan is *progressing* if and only if any instance of any action is used at most once in the plan.

The notion of progressing is motivated in a similar way as the use of protocol roles in [18]. The idea is that whenever one step of a protocol is performed, one never needs to repeat this step again. While the progressing condition naturally appears in the specification of security of protocols, note that in Section 4 we differ from [18] when we represent memory bounded intruder and use only balanced actions. In particular, in [18], the intruder can copy facts, *i.e.*, the intruder's memory is unbounded.

For another example, when a process has a deadline, such as a scientific paper submission, one must complete it on time, before some deadline, otherwise the paper is not accepted for a review. The use of deadlines is another form of bounding processes: since actions take some positive time to be performed, it must be the case that the whole operation is completed within a bounded number of transactions.

Grant Proposal Scenario. We now specify an example of a progressing collaborative process, called *grant proposal*, where a leader is involved. There, different agents or researchers collaborate to write a proposal, which includes a budget and a technical text. Among the agents, we distinguish a leader called PI (principal investigator) and we call the remaining coPIs (co-principal investigators). The PI is responsible for sending the complete project to the funding agency and for coordinating the coPIs. The task is to find a plan so that all coPIs finish writing their part of the text and budget and send them to the PI well before the deadline of the proposal, so that the PI can wrap up and send the final project to the funding agency. The critical states of this example is any state where the time has passed the deadline.

Some of the actions belonging to a coPI are depicted in Figure 1.1. At the beginning of a collaboration no coPI has a budget (*noBudget*). At some point, a coPI starts working on the proposal by requesting his accounting office to write the budget (*writeBudget*). This requires *TW* time units, as specified by the second action. By the third action, the office sends the budget to the coPI. At this point, however, if the coPI does not know the title of the project (*no title*), he cannot send the final proposal to the Dean's office (*uni*) for final approval. Only when the title is made public by the PI, a coPI can do so and the Dean's office requires TU time units to approve the budget. These are specified by the fourth and fifth actions. Finally, when the budget is approved, the budget is made available only to the PI by using the private channel to the leader Al. Here, time is another agent of the system whose unique action is to move time forward by incrementing the value in the fact $Time(t)$.

There are more actions to the scenario, such as the action where a coPI can also revise the budget and send it back to his accounting office for modifications, as well as the actions of the PI. Since the budget of one coPI is normally not available to the other coPIs, we consider that in the process of writing the budgets, coPIs do not communicate among themselves, but only with the leader, PI, and their internal organizations, e.g., accounting offices.

It is easy to check that the scenario described above is progressing. Each action corresponds to checking a box, that is, once it is performed, it is not repeated. This is enforced in the rules by using the time agent who moves time forward. Since actions either require some time, *e.g.*, TW and TU, to be performed or necessarily occur after another action is performed, *e.g.*, the coPI can only send the budget when the PI has made the title public, an instance of an action can never be repeated. For instance, if the example above is extended so that the coPI and his office send several versions of the budget back and forward, with revisions, all these actions are different instances of the same rules each with a different time value. Moreover, since time is discrete and all actions need to be performed until a deadline is reached, there cannot be infinitely many revisions. Hence, any computation run in this system is bounded. The progressing nature of the process is syntactically emphasized by predicates P_i which represent the phase of the process. Actions allow agents only to move forward to the next phase.

$$\begin{array}{rcl}
Time(t) & \rightarrow_{time} & Time(t+1)\\
Time(t), Al(noBudget, A) & \rightarrow_A & Time(t), P_0(A, office, writeBudget, t+TW)\\
Time(t), P_0(A, office, writeBudget, t) & \rightarrow_A & Time(t), P_1(A, coPI, no\ title, budget)\\
Pub(title), P_1(A, coPI, no\ title, budget) & \rightarrow_A & Pub(title), P_2(A, coPI, title, budget)\\
Time(t), P_2(A, coPI, title, budget) & \rightarrow_A & Time(t), P_3(A, uni, title, budget, t+TU)\\
Time(t), P_3(A, uni, title, budget, t) & \rightarrow_A & Time(t), Al(budget, A)
\end{array}$$

Figure 1.1: The set of actions involving the writing of a budget for a coPI called A and of the agent time. Here TW and TU are, respectively, the time needed for A's accounting office to write a budget and for the Dean of A's university to approve the final budget.

With respect to confidentiality issues, this is a scenario where weak plan compliance is best suited. The problem is to determine whether there exists a plan which allows the researchers to write the grant proposal before the deadline.

1.1.4 The Computational Problems

Now that we have introduced all the relevant definitions we can state the computational problems for which we would like to determine the computational complexity. They can each be considered special cases of the following general problem.

The Collaborative Planning Problem with Confidentiality. Let T be a local state transition system with a finite set of actions $\mathcal{T}$, in initial configuration W, with a finite set of goals G, and a finite set of critical configurations C. Determine whether T has a plan leading from W to one of the goals in G, which complies with all the agents confidentiality policies.

We will identify some restrictions to the actions of local state transition systems. These restrictions are useful to obtain classes of LSTSes for which solving the policy compliance problems becomes feasible. For instance, if no restrictions are made, one can show that weak plan compliance is undecidable [26], even if actions do not create nonces.

We can make two independent choices. Firstly, we must choose which definition of policy compliance we would like to use.

- (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?

[1]$M_?(s)$ stands for any memory fact of the intruder such as $M_n(s)$ and $M_k(s)$

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'.

Second, we choose the type of actions used in the system. This yields different computational problems. We will mostly consider balanced systems. In particular, we will consider LSTSes with monadic, and, hence, balanced rules. In balanced systems the size of configurations contained in a plan is always the same as in the initial configuration. Specifically, for systems with only monadic actions, whenever an action is used, only one fact is changed. As discussed in [26], the restriction of balanced actions provides decidability of weak plan compliance. On the other hand, the restriction of monadic actions is new and it will be explored later in this paper.

Additionally, we can consider the above problems in relation to progressing behavior. In [26, 27], plans were allowed to use an instance of an action as many times needed. Here, however, we can accommodate the assumption that a collaborative system is progressing and consider following problems:

- (Progressing 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 progressing plan which leads from W to Z?

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

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

- (Progressing 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 progressing 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?

- (Strictly progressing 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 progressing compliant plan which leads from W to Z such that for each agent A_i and for each configuration Y along the plan, whenever $Y \rhd^*_{-A_i} V$ in progressing sequence of actions, then V is not critical for A_i?

- (Secrecy problem for progressing plans) Is there a progressing plan from the initial configuration to a configuration in which the adversary M owns the fact $M_?(s)$, where s is a secret originally owned by another participant?

The strictly progressing and progressing versions of system and plan compliance problems differ in how the critical configurations may be reached. While the latter problem requires that critical configurations cannot be reached by any plan, the former considers only progressing plans. The secrecy problem for progressing plans is the same as the secrecy problem but it is restricted to progressing plans only. Since all players in the system have the same capabilities, if we assume that the system is progressing, then so is the adversary, that is, the adversary is also not allowed to repeat an instance of any action.

1.2 Connections to Linear Logic

In this section, we provide a precise semantics for LSTSes with actions that can create nonces in the terms of linear logic [20], as is done in [27, 11]. However, here we obtain a tighter correspondence by using the notion of focusing introduced by Andreoli in [3].

Firstly, we review some of linear logic's basic proof theory. *Literals* are either atoms or their negations. The connectives $\otimes$ and $⅋$ and the units 1 and $\bot$ are *multiplicative*; the connectives $\&$ and $\oplus$ and the units $\top$ and 0 are *additive*; $\forall$ and $\exists$ are (first-order) quantifiers; and $!$ and $?$ are the *exponentials*. We assume that all formulas are in *negation normal form*, that is, negation has atomic scope.

We encode a rule of the form $X_A X_{Al} X_{pub} \to_A \exists \vec{t}. Y_A Y_{Al} Y_{pub}$ as the linear logic formula $\forall \vec{x}[\bigotimes\{q_A X_A X_{Al} X_{pub}\} \multimap \exists \vec{t}. \bigotimes\{q_A Y_A Y_{Al} Y_{pub}\}]$, where $\vec{x}$ are the free variables appearing in the rule and where the atomic formula q_A is used only to mark that this action belongs to agent A. Moreover, the encoding of a set of transition rules $\ulcorner R_T \urcorner$ is the set with the encoding of all the transition rules in R_T, and the set of propositions used to mark a rule to an agent is defined as $Q_I = \{q_A : A \in I\}$. One feature of this particular encoding is that the creation of nonces is specified by using standard quantifiers.

The proof of the correspondence relies on the completeness theorem of the focused proof system for linear logic, LLF, depicted in Figure 1.2. LLF is a one-sided version of linear logic, where sequents of the form $\Gamma \vdash \Delta$ appear instead in the form $\vdash \Gamma^\perp, \Delta$. Moreover, the structural rules are not explicitly mentioned in the system, but instead they are incorporated in the introduction rules. This is accomplished by distinguishing unbounded and bounded formulas into two different contexts $\Theta : \Gamma$ to the left of the $\Uparrow$ and $\Downarrow$, where Θ contains the multiset of unbounded formulas and Γ the multiset of bounded formulas. For example, in the tensor introduction rule, while the bounded context is split among the premises, the unbounded context is copied to both premises.

In order to introduce LLF, we first classify the connectives $1, \otimes, \oplus$, and $\exists$ as positive and the remaining as negative. This distinction is natural as the introduction rules for the positive connectives are not-necessarily invertible, while the rules for the negative connectives are invertible. The same distinction, however, does not apply so naturally to literals and hence these are *arbitrarily* classified as positive or negative. Positive polarity literals and formulas whose main connective is positive are classified as positive formulas and the remaining as negative formulas. There are two different sequents in LLF: those containing $\Uparrow$ which belong to the negative phase where only negative formulas are introduced, and those containing $\Downarrow$ which belong to the positive phase and only positive formulas are introduced.

A focused proof is a normal-form proof for proof search. Inference rules that are not necessarily invertible are classified as positive, and the remaining rules as negative. Using this classification, focused proof systems reduce proof search space by allowing one to combine a sequence of introduction rules of the same polarity into larger derivations, which can be seen as “macro-rules”.

Introduction Rules

$$\frac{\vdash \Theta : \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow L, \bot}\,[\bot] \qquad \frac{\vdash \Theta : \Gamma \Uparrow L, F, G}{\vdash \Theta : \Gamma \Uparrow L, F \parr G}\,[\parr] \qquad \frac{\vdash \Theta, F : \Gamma \Uparrow L}{\vdash \Theta : \Gamma \Uparrow L, ?F}\,[?]$$

$$\frac{}{\vdash \Theta : \Gamma \Uparrow L, \top}\,[\top] \qquad \frac{\vdash \Theta : \Gamma \Uparrow L, F \quad \vdash \Theta : \Gamma \Uparrow L, G}{\vdash \Theta : \Gamma \Uparrow L, F \,\&\, G}\,[\&] \qquad \frac{\vdash \Theta : \Gamma \Uparrow L, F[c/x]}{\vdash \Theta : \Gamma \Uparrow L, \forall x\, F}\,[\forall]$$

$$\frac{}{\vdash \Theta : \Downarrow 1}\,[1] \qquad \frac{\vdash \Theta : \Gamma \Downarrow F \quad \vdash \Theta : \Gamma' \Downarrow G}{\vdash \Theta : \Gamma, \Gamma' \Downarrow F \otimes G}\,[\otimes] \qquad \frac{\vdash \Theta : \Uparrow F}{\vdash \Theta : \Downarrow !\,F}\,[!]$$

$$\frac{\vdash \Theta : \Gamma \Downarrow F}{\vdash \Theta : \Gamma \Downarrow F \oplus G}\,[\oplus_l] \qquad \frac{\vdash \Theta : \Gamma \Downarrow G}{\vdash \Theta : \Gamma \Downarrow F \oplus G}\,[\oplus_r] \qquad \frac{\vdash \Theta : \Gamma \Downarrow F[t/x]}{\vdash \Theta : \Gamma \Downarrow \exists x\, F}\,[\exists]$$

Identity, Reaction, and Decide rules

$$\frac{}{\vdash \Theta : A_p^{\perp} \Downarrow A_p}\,[I_1] \qquad \frac{}{\vdash \Theta, A_p^{\perp} : \Downarrow A_p}\,[I_2] \qquad \frac{\vdash \Theta : \Gamma, S \Uparrow L}{\vdash \Theta : \Gamma \Uparrow L, S}\,[R \Uparrow]$$

$$\frac{\vdash \Theta : \Gamma \Downarrow P}{\vdash \Theta : \Gamma, P \Uparrow}\,[D_1] \qquad \frac{\vdash \Theta, P : \Gamma \Downarrow P}{\vdash \Theta, P : \Gamma \Uparrow}\,[D_2] \qquad \frac{\vdash \Theta : \Gamma \Uparrow N}{\vdash \Theta : \Gamma \Downarrow N}\,[R \Downarrow]$$

Figure 1.2: The focused proof system, LLF, for linear logic [3]. Here, L is a list of formulas, Θ is a multiset of formulas, Γ is a multiset of literals and positive formulas, A_p is a positive literal, N is a negative formula, P is not a negative literal, and S is a positive formula or a negated atom.

For example, the macro-rule obtained from (focusing on) the encoding of a transition rule $X_A, X_{Al} X_{pub} \to_A \exists \vec{t}. Y_A, Y_{Al} Y_{pub}$ can be made to match the following “macro-rule”:

$$\begin{array}{c} \Gamma, q_A, Y_A\sigma, Y_{Al}\sigma, Y_{pub}\sigma \vdash C \\ \hline\hline \Gamma, q_A, X_A\sigma, X_{Al}\sigma, X_{pub}\sigma \vdash C \end{array}$$

where σ is the substitution used to trigger the rule and where new eigenvariables are created when introducing the existential quantifiers appearing in the encoding of the rule. Once the goal is reached, that is, the bounded context to the left of the $\vdash$ contains all facts in the partial goal, R, we finish the proof by using the linear logic formula $\top$ to consume all other facts, Y, that are not mentioned in the partial goal, as illustrates the following derivation (focused on the formula in the right-hand-side):

$$\dfrac{\dfrac{}{!\ulcorner R_T \urcorner, Q_I \vdash \bigotimes\{Q_I\}} \quad \dfrac{}{!\ulcorner R_T \urcorner, R \vdash \bigotimes\{R\}} \quad \dfrac{}{!\ulcorner R_T \urcorner, Y \vdash \top}}{!\ulcorner R_T \urcorner, Q_I, Y, R \vdash \bigotimes\{Q_I, R\} \otimes \top}$$

Theorem 1.15. *Let $T = \langle \Sigma, I, l, R_T \rangle$ be a local transition system. Let W and R be two states under the signature Σ. Then the sequent $!\ulcorner R_T \urcorner, Q_I, W \vdash \bigotimes\{Q_I, R\} \otimes \top$ is provable in linear logic iff $W \rightsquigarrow_T^* R$.*

Proof Consider the encoding of the transition rule $X_A, X_{Al}, X_{pub} \rightarrow_A \exists \vec{t}. Y_A, Y_{Al}, Y_{pub}$:

$$F = \forall \vec{x}[\bigotimes\{q_A X_A X_{Al} X_{pub}\} \multimap \exists \vec{t} \bigotimes\{q_A Y_A Y_{Al} Y_{pub}\}]$$

which is on the left-hand-side of the sequent and therefore we use its negation in the one-sided LLF system:

$$F^{\perp} = \exists \vec{x}[\bigotimes\{q_A X_A X_{Al} X_{pub}\} \otimes \forall \vec{t} \parr^{2}\{q_A^{\perp} Y_A^{\perp} Y_{Al}^{\perp} Y_{pub}^{\perp}\}].$$

Assume now that all atomic formulas have positive polarity, and consequently their negation negative polarity. The focused derivation introducing $F^{\perp}$ has to be necessarily of the form:

$$\dfrac{\dfrac{\dfrac{\vdash \Theta : \Delta \Downarrow \bigotimes\{q_A X_A X_{Al} X_{pub}\} \qquad \vdash \Theta : \Gamma \Downarrow \forall \vec{t} \parr\{q_A^{\perp} Y_A^{\perp} Y_{Al}^{\perp} Y_{pub}^{\perp}\}}{\vdash \Theta : \Gamma, \Delta \Downarrow \bigotimes\{q_A X_A X_{Al} X_{pub}\} \otimes \forall \vec{t} \parr\{q_A^{\perp} Y_A^{\perp} Y_{Al}^{\perp} Y_{pub}^{\perp}\}}\,[\otimes]}{\vdash \Theta : \Gamma, \Delta \Downarrow F^{\perp}}\,[n \times \exists]}{\vdash \Theta : \Gamma, \Delta \Uparrow \cdot}\,[D_2]$$

Since $\otimes$ is a positive connective, the left-premise is necessarily introduced by a completely positive phase introducing all tensors in $\bigotimes(q_A X_A X_{Al} X_{pub})$ until one is focused only on atomic formulas. Since atomic formulas are positive, the only applicable rule at this point is an initial rule. This forces Δ to be exactly the negation of the facts in the pre-condition of the transition rule union the fact $q_A^{\perp}$. In contrast, since $\forall$ and $\parr$ are negative connectives, the right-premise is necessarily introduced by a negative phase introducing these connectives. Hence, the macro-rule introducing an encoding of the transition rule is necessarily of the form:

$$\dfrac{\vdash \Theta : \Gamma, q_A^{\perp}, Y_A^{\perp}\sigma, Y_{Al}^{\perp}\sigma, Y_{pub}^{\perp}\sigma \Uparrow \cdot}{\vdash \Theta : \Gamma, q_A^{\perp}, X_A^{\perp}\sigma, X_{Al}^{\perp}\sigma, X_{pub}^{\perp}\sigma \Uparrow \cdot}$$

[2] If you have problems seeing this symbol (big par), please use the CMLL fonts available at `http://iml.univ-mrs.fr/~beffara/soft/`.

Notice that if the pre-condition of the rule is not satisfied, then there is no focused proof which focuses on the encoding of this transition. This handles the inductive case.

The base case consists in checking if a partial goal is reached. This is specified in a similar way by the “macro-rule” introducing the formula $\bigotimes\{R, Q_I\} \otimes \top$:

$$\dfrac{\dfrac{\overline{\overline{\vdash \Theta : \Delta \Downarrow \bigotimes\{R, Q_I\}}} \quad \dfrac{}{\vdash \Theta : \Gamma \Downarrow \top}\,[R \Downarrow, \top]}{\vdash \Theta : \Gamma, \Delta \Downarrow \bigotimes\{R, Q_I\} \otimes \top}\,[\otimes]}{\vdash \Theta : \Gamma, \Delta \Uparrow \cdot}\,[D_2]$$

The focusing discipline forces that Δ contains exactly the negation of the facts appearing in R and Q_I and that Γ are the remaining facts in the state.

At the end, we obtain a one-to-one correspondence between focused proofs and runs of the encoded local state transition system. □

The use of the $\top$ is enough to specify partial goals in linear logic, obtaining hence the same effect as in [27] where they used instead *affine linear logic*. However, since there is no clear focused proof system for affine linear logic, we prefer the encoding in linear logic.

In the encoding above, we have not captured the operational semantics of LSTSes with progressing behavior. In order to do so, one would need to remember the instances of the rules that were used before and disallow them to be used again. A logical specification of such operation does not seem to be trivial and is left for future work.

Chapter 2

Formalizing Freshness for LSTSes with Balanced Actions

In principle a plan can be exponentially long. Moreover, since actions in a plan can create fresh values such exponentially long plans may involve exponentially many fresh values. In Section 2.1 we provide examples of such plans including a solution for the Towers of Hanoi puzzle suitably modified to include nonce creation, which would, in principle, introduce an exponential number of fresh values in the plan. Such plans, and in particular the use of an exponential number of fresh values, in principle seems to preclude PSPACE membership of the secrecy problem and all compliance problems given in Section 1.1.4.

In Section 2.2 we show how to circumvent this problem by reusing obsolete constants instead of creating fresh values. We show that one only requires a small number of nonces in a plan.

2.1 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 and an exponential attack of a simple protocol introduced in [18]. Such plans seem to preclude PSPACE membership, especially when nonces are involved, since there can bean an a priori exponential number of nonces in such plans.

2.1.1 Towers of Hanoi

Towers of Hanoi is a well-known mathematical game or puzzle. It consists of three pegs b_1, b_2, b_3 and a number of disks a_1, a_2, a_3, $\dots$ of different sizes which can slide onto any peg. The puzzle starts with the disks neatly stacked in ascending order of size on one rod, 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:

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

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

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

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

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

Finally, the only action in our system is:

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

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 the previous "ticket" with a fresh identifier:

$$\begin{aligned} &T(t)\ Clear(x)\ On(x,y)\ Clear(z)\ S(x,z) \to \\ &\qquad\qquad \exists t'.T(t')\ Clear(x)\ Clear(y)\ On(x,z)\ S(x,z) \end{aligned}$$

so that $T(t)$ means that the current value is t. As before, given n disks, all plans must be of the exponential length 2^n-1, at least. Consequently, within the modified version, a straightforward plan includes, a priori, an exponential number of nonces.

2.1.2 Exponential Protocol Anomaly

Protocols and the relevant security problems have been modeled with multiset rewriting rules, see for example [18]. We are therefore able to apply our LSTSes to protocol security analysis, as we do in Sections 4 and 5. There are protocols and the respective anomalies or attacks that involve an exponential number of sessions, *i.e.*, an exponential number of actions needs to be applied. In the Section 5.5 we encode such a protocol, originally introduced in [18], using LSTSes with balanced actions.

This protocol, informally described in Figure 2.1, 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. 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. It has the side-effect of broadcasting the SECRET if C receives the error message.

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 2.1: Audited Key Distribution Protocol for $s = 4$

It is easy to check that when a Dolev-Yao intruder is present, he can discover the secret in an exponential number of steps, assuming he does not possess the key K. In particular, he routes the initial message $\{0, 0, 0, 0\}_K$ from the server A through $2^s - 1$ sessions. In each session, the value of the encrypted binary number is increased by one and after $2^s - 1$ sessions the encrypted messages contains only 1's. This message is then sent to C, and causes the broadcast of the SECRET unencrypted. The intruder, therefore, learns the secret using an exponentially long plan.

2.2 Guarded Nonce Creation

In this section we show how to circumvent the problem of possibly having an exponential number of nonces in a plan by reusing obsolete constants instead of updating with fresh values. We show that one only requires a small number of nonces in a plan.

There are two main views on fresh values, one is quite strict and the other is more local. For example, when an artist creates his original work of art, it is different to any other song, sculpture or painting made in human history. However, a fresh value can also be seen as a local object, that is a fresh value can be considered as any value that does not belong to any agent of the system in a particular configuration or at a particular moment or period of time. Under the latter interpretation, even values that appeared before in a plan, but that do not appear in a configuration anymore, can be considered fresh.

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

We 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. Namely, there are at most mk occurences of constants (including nonces) in any configuration of m facts of the size k. We can imagine them as memory slots. Therefore, any balanced action that involves nonces, inserts nonces into the memory providing there is an empty slot available, or replaces a stored value with a fresh one, different to any of mk values stored in system memory at that moment.

Consequently, in a given planning problem we only need to consider a small number of *fresh* nonces, which can be fixed in advance. This is formalized by the following theorem:

Theorem 2.1. *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 2.1 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 signature:

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

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 [13], we say that these configurations above are α-equivalent.

Definition 2.2. 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 [32].

Lemma 2.3. 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, and we say r' is an action of *guraded 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 2.1:

Proof (of Theorem 2.1). The proof is by induction on the length of a plan and it is based on Lemma 2.3. Let T be an LSTS with balanced actions that can create nonces, m the number of facts in a 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 $\mathcal{N}_{m,k}$.

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 2.3 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 2.4. For LSTSes with balanced actions that can create nonces, we only need to consider the planning problem with a polynomial number of *fresh* values, which can be fixed in advance, with respect to the number of facts in the initial configuration and the upper bound on the size of facts.

Notice that, since plans can be of exponential length, a nonce name from $\mathcal{N}_{m,k}$ can, in principal, be used in guarded nonce creation an exponential number of times.

Chapter 3

Complexity Results

In this Section we discuss complexity results for different problems introduced in Section 1.1.4, namely, the weak plan compliance problem, the plan compliance problem, the system compliance problem and the secrecy problem.

3.1 Complexity of Some Problems without Nonces

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 [18, Proposition 5.4].

Theorem 3.1. *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 [27, 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 weak plan compliance problem for LSTSes with monadic actions is 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.

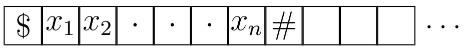
Theorem 3.2. *Given an LSTS with only actions of the form $ab \to a'b$, then the problems of weak plan compliance, plan compliance, system compliance and the secrecy problem are PSPACE-hard.*

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

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 $x_1x_2\dots x_n$, is written in cells 1, 2,…, n on the tape. In addition, a special marker # is written in the $(n+1)$-th cell.



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

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

For each n, we design a local state transition system T_n as follows:
Firstly, we introduce the following propositions: $R_{i,\xi}$ which denotes that *"the i-th cell contains symbol ξ"*, where $i = 0, 1, \dots, n+1$, 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 $j=0, 1, \dots, n+1$, 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 $\xi_0\xi_1\xi_2\dots\xi_i\dots\xi_n\xi_{n+1}$ 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):

$$S_{j,q}R_{0,\xi_0}R_{1,\xi_1}R_{2,\xi_2}\cdots R_{n,\xi_n}R_{n+1,\xi_{n+1}}. \tag{3.1}$$

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:

$$\begin{array}{lll} S_{i,q}R_{i,\xi} \to_A F_{i,\gamma}R_{i,\xi}, & F_{i,\gamma}R_{i,\xi} \to_A F_{i,\gamma}H_{i,\gamma}, & F_{i,\gamma}H_{i,\gamma} \to_A G_{i,\gamma}H_{i,\gamma}, \\ G_{i,\gamma}H_{i,\gamma} \to_A G_{i,\gamma}R_{i,\eta}, & G_{i,\gamma}R_{i,\eta} \to_A S_{i_D,q'}R_{i,\eta}, & \end{array} \tag{3.2}$$

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 $x_1 x_2 \ldots x_n$, 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. Namely, because of the specific form of our actions in (3.2), 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 (3.2) 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 (3.1), 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 $x_1 x_2 \ldots x_n$ 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 (3.2) 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.

3.2 Complexity in Systems with 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 3.2, we can infer that this problem is indeed PSPACE-complete.

Recall that, in Section 2.2 we introduce a formalization of freshness in balanced systems. Instead of (proper) nonce generation, in balanced systems we consider *guarded nonce generation*, see Lemma 2.3. We are then able to simulate plans that include actions of nonce generation 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 signature.

Lemma 3.3. Given an LSTS T under a finite signature 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 signature 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 signature Σ or nonce names.

However, following Theorem 2.1, 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 signature Σ. □

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 2.1, the number of actions in the plan may already be exponential in the size of the inputs precluding PSPACE membership of plan generation.

Moreover, these plans could, in principle, also involve an exponential number of nonces, as discussed at the end of Section 2. For the reason above we follow [27] and use the notion of "scheduling" a plan in which an algorithm will also take an input i and output the i-th step of the plan.

Definition 3.4. 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 [27], we assume that when given an LSTS, there are three programs, $\mathcal{C}, \mathcal{G}$, and $\mathcal{T}$, such that they return the value 1 in polynomial space when given as input, respectively, a configuration that is critical, a configuration that contains the goal configuration, and a transition that is valid, that is, an instance of an action in the LSTS, and return 0 otherwise.

Theorem 3.5. *Given an LSTS with balanced actions that can create nonces, then the weak plan compliance problem and the secrecy problem are in PSPACE.*

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

We first prove that the weak plan compliance problem is in PSPACE. We modify the algorithm proposed in [27] in order to accommodate the updating of nonces. The algorithm must return "yes" whenever there is compliant plan from the initial configuration W to a goal configuration. In order to do so, we construct an algorithm that searches non-deterministically whether such configuration *is reachable*, that is, a configuration S such that $\mathcal{G}(S) = 1$. Then we apply Savitch's Theorem to determinize this algorithm.

The algorithm begins with $W_0 := W$. For any $t \geq 0$, we first check if $\mathcal{C}(W_t) = 1$. If this is the case, then the algorithm outputs "no". We also check whether the configuration W_t is a goal configuration, that is, if $\mathcal{G}(W_t) = 1$. If so, we end the algorithm by returning "yes". Otherwise, we guess a transition r such that $\mathcal{T}(r) = 1$ and that is applicable using the configuration W_t. If no such action exists, then the algorithm outputs "no". Otherwise, we replace W_t by the configuration W_{t+1} resulting from applying the action r to W_t. Following Lemma 3.3 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.

In order to accommodate nonce creation, we use guarded nonce creation. This is done, as in the proof of Theorem 2.1, by replacing the relevant nonce occurrence(s) with nonce names from a fixed set of nonce names so that they are different from any of the nonces in the enabling configuration.

We now show that this algorithm runs in polynomial space. We start with the step-counter: The greatest number reached by this counter is $L_T(m,k)$. 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 2.1 there are at most polynomially many nonces contained in any plan, 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. □

In order to show that the system compliance problem for LSTS-es that can create nonces is in PSPACE we modify the algorithm proposed in [27] to accommodate the guarded nonce creation.

Theorem 3.6. *Given an LSTS with balanced actions that can create nonces, then the system compliance problem is in PSPACE.*

Proof Again we rely on the fact that NPSPACE, PSPACE, and co-PSPACE are all the same complexity class. We use the same notation from the proof of Theorem 3.5 and make the same assumptions. Following Theorem 2.1 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 3.3 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 3.5, 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 3.5, 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. As we have already pointed out, the plan compliance problem can be re-stated as a weak plan compliance problem which considers critical both the original critical configurations of the system as well as the semi-critical configurations of any agent. 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. Recall that a configuration is semi-critical for an agent A if a configuration critical for A can be reached by other participants of the system without the participation of A.

Theorem 3.7. *Given an LSTS with balanced actions that can create nonces, then the plan compliance problem is in PSPACE.*

Proof The proof is similar to the proof of Theorem 3.5 and the proof of the PSPACE result of the plan compliance for balanced systems in [35]. 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 3.5. 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.

We start by constructing 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 3.5 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 2.1 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 3.5 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 3.5 to conclude that the plan compliance problem is in PSAPCE. $\Box$

Given the PSPACE lower bound for the secrecy, weak plan compliance, system compliance, and the plan compliance problem in Theorem 3.2 and the PSPACE upper bound given in the theorems above, we can conclude that all these problems are PSPACE-complete.

Table 3.1 summarizes the complexity results for the compliance and secrecy problems.

Discussion on related work This PSPACE-complete result contrasts with results in [18], where the secrecy problem is shown to be undecidable. Although in [18] 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 [18], a constant bound on the number of nonces that can be created was imposed, whereas such a bound is not imposed here.

Table 3.1: Summary of the complexity results for the weak compliance and the secrecy problems. We mark the new results appearing here with a $\star$. The complexity for the system compliance problem when actions are possibly unbalanced and can create fresh values is shown to be undecidable in Section 3.4.2.

Compliance Problems	Balanced Actions: No fresh values	Balanced Actions: Possible fresh values	Possibly unbalanced actions and no fresh values
Secrecy	PSPACE-complete [27]	PSPACE-complete*	Undecidable [18]
Weak Plan	PSPACE-complete [27]	PSPACE-complete*	Undecidable [26]
System	PSPACE-complete [27]	PSPACE-complete*	EXPSPACE-complete [26]
Plan	PSPACE-complete [27, 35]	PSPACE-complete*	Undecidable [26]

3.3 Complexity in Balanced Progressing Systems

Now, we investigate the complexity of the progressing weak plan compliance problem when one is allowed to use only monadic actions that cannot create nonces. These restrictions reflect the assumptions discussed in the Introduction, namely, that many systems have a constant number of names and that they have a progressing behavior. For instance, the grant proposal example introduced in Section 1.1.3 is in this class of LSTSes. We show that this problem is NP-complete.

Theorem 3.8. *Given a local state transition system with only monadic actions of the form $ab \to a'b$, the progressing weak plan compliance problem, the progressing plan compliance problem and the secrecy problem for progressing plans are NP-complete when the size of facts is fixed.*

Proof We start by proving the NP-hard lower bound for the partial reachability problem. In our proof, we do not use critical states nor function symbols, and use only one single agent. In particular, we reduce the 3-SAT problem which is well-known to be NP-complete [16] to the problem of reachability using transition rules and a bounded number of state variables. Consider the formula below in conjunctive normal form with three variables per clause, classified as 3-CNF, to be the formula for which one is interested in finding an model: $C = (l_{11} \lor l_{12} \lor l_{13}) \land \cdots \land (l_{n1} \lor l_{n2} \lor l_{n3})$, where each l_{ij} is either an atomic formula v_k or its negation $\neg v_k$. We encode the 3-SAT problem by using two sets of rules. The first one builds an interpretation for each variables, v_k, appearing in C as follows

$$v_k \to_A t_k \qquad \text{and} \qquad v_k \to_A f_k,$$

where the first rule rewrites the variable v_k to true, denoted by t_k, and the second to false, denoted by f_k. The second set of rules checks if the formula C is satisfiable given an interpretation:

$$\begin{array}{ll}
S_{(v_k \lor l_{j2} \lor l_{j3}) \land C}, t_k \to_A S_C, t_k & S_{(\neg v_k \lor l_{j2} \lor l_{j3}) \land C}, f_k \to_A S_C, f_k \\
S_{(l_{j1} \lor v_k \lor l_{j3}) \land C}, t_k \to_A S_C, t_k & S_{(l_{j1} \lor \neg v_k \lor l_{j3}) \land C}, f_k \to_A S_C, f_k \\
S_{(l_{j1} \lor l_{j2} \lor v_k) \land C}, t_k \to_A S_C, t_k & S_{(l_{j1} \lor l_{j2} \lor \neg v_k) \land C}, f_k \to_A S_C, f_k
\end{array}$$

where $1 \leq j \leq n$ and v_k is a variable appearing in C. Thus, we have a total of $(2 \times m + 6 \times n)$ rules and a total of $(3 \times m + n + 1)$ state variables, where m and n are, respectively, the number of variables and clauses in C.

Given a 3-CNF formula C and the set of rewrite rules, R_T, shown above, we prove soundness and completeness as shown below.

($\Rightarrow$) If C is satisfiable, we rewrite the variables in C according to the model of C by using the first set of rules in R_T and then check for satisfiability using the second set of rules in R_T.

($\Leftarrow$) It is similar to the previous direction. A state containing $S_\emptyset$ is reached only if a partial interpretation for the variables in C is build by using the first set of rules in R_T. This interpretation can be completed by assigning, for example, false to all other variables and will be necessarily a model of C.

For the NP upper bound, we propose the following algorithm:

Let $T = \langle \Sigma, I, l, R_T \rangle$ be a local transition system, such that n is the number of rules in R_T, d is the number of constant and function symbols in Σ, k_1 the upper bound on the size of facts, where the size of a fact is total number (including repetitions) of function and constant symbols in it, and k_2 the upper bound on the number of different variables appearing in a rule. Here we assume k_1 and k_2 to be much smaller than d. Since facts are bounded, we do not need to consider terms that have a size greater than k_1; therefore we need to consider at most d^{k_1} terms. Since, in progressing plans, one is allowed to use only one instance of a rule, the length of runs is bounded by $n \times (d^{k_1})^{k_2} = n \times d^{k_1 k_2}$, which is polynomial on the number of rules and symbols. Assume that W is the initial configuration and Z the goal configuration and that one can check in polynomial time whether a configuration is critical or not.

We show below that there is a polynomial-time algorithm that checks for valid computation runs.

Let S_i be the state at step i, so $S_0 = W$; R_i be the set of pairs, $\langle r, \sigma \rangle$, of rules and substitutions used before step i, so $R_0 = \emptyset$; and let the following rule, $r_i : ab \to a'b$, be used with the substitution σ_i at step i

1. Check if $Z \subseteq S_{i-1}$, then ACCEPT; otherwise continue;
2. Check if Z is not a critical state, then continue; otherwise FAIL;
3. Check if $a\sigma_i, b\sigma_i \in S_{i-1}$, then continue; otherwise FAIL;
4. Check if $\langle r_i, \sigma_i \rangle \in R_{i-1}$, then FAIL; otherwise continue;
5. $S_i = S_{i-1} \cup \{a'\sigma_i\} \setminus \{a\sigma_i\}$;
6. $R_i = R_{i-1} \cup \{\langle r_i, \sigma_i \rangle\}$;
7. Increment i.

Since the size of facts is bounded, all steps are done in polynomial time. The only step that may not be apparent is step 3. However, the set R_i is bounded by i, that is, by the length of the computation run. Therefore, the reachability problem is in NP. $\square$

Notice that the above upper bound works even if actions are not monadic. It even works if we assume that actions can create a bounded number, n, of nonces. One would only need to extend the number of constants by n and the algorithm would work in a similar way, as states the following proposition:

Proposition 3.9. Given a local transition system, T, such that all its rules are of the form $a, b \to \exists \vec{x}.a', b$, then the progressing weak plan compliance problem, the progressing plan compliance problem and the secrecy problem for progressing plans are in NP when only a bounded number of nonces are allowed and when the size of facts is fixed.

Proof Let $T = \langle \Sigma, I, l, R_T \rangle$ be a local transition system, such that n is the number of rules in R_T, d is the number of constants in Σ, k_1 the upper bound on the size of facts, where the size of a fact is total number (including repetitions) of function and constant symbols in it, k_2 the upper bound on the number of different variables appearing in a rule, and m be the bound on the number of nonces allowed. Following [18], we can assume that all nonces are created before hand extending, hence, the number of constants to $d + m$. Thus the length of computation is bounded by $n \times (d + m)^{k_1 k_2}$. Moreover we remove the existential quantifiers of all rules. Then the proof follows in the same way as in Theorem 3.8. $\square$

The NP upper bound obtained above relies on the progressing fact, since this bounds the length of a computation run. This is different from the NP upper bound obtained [2, 36] in the context of protocol security. In their models, the progressing condition is incorporated syntactically into the rules of the theories. Specifically, they use role predicates of the form A_i contain an index i denoting the stage in the protocol a principal is: once a stage of one instance of a protocol role is reached it is never repeated again. The NP-complete result in [2, 36] is obtained by further restricting systems to have only a bounded number of roles. This implicitly bounds the signature used to be finite. As shown in our proof, this restriction, together with the implicit progressing behavior of protocols, is enough to ensure that runs have a polynomial length and from there, show that the reachability problem is in NP.

Interestingly, the NP-hardness result obtained above is replaced by a PSPACE-hardness result, see Theorem 3.2, if we allow actions to be used as many times needed, even when LSTSes are restricted to have only monadic actions that do not create nonces. This shows that the notion of progressing is indeed important to guarantee the lower complexity. This result also improves the result in [27, Theorem 6.1] since in their encoding they allowed any balanced actions including non-monadic ones, whereas here we use only monadic actions. The main challenge here is to simulate operations over a non-commutative structure (tape) by using a commutative one (multiset).

We now return to the problem of progressing weak plan compliance, where an instance of an action can be used at most once, but now, we allow monadic actions to create nonces. That is, we trade the absence of the progressing condition for the ability to create nonces. It turns out that such problem is also PSPACE-hard. Therefore, the progressing condition alone is not enough to guarantee an NP complexity, but one also needs to forbid the creation nonces.

Theorem 3.10. *Given a local state transition system with only monadic actions of the form* $ab \to \exists \vec{x}.a'b$, *the progressing weak plan compliance problem, the progressing plan compliance problem and the secrecy problem for progressing plans are PSPACE-hard.*

Proof (Sketch) The proof goes in a similar fashion as the lower bound proof of Theorem 3.2. However, we cannot use the same encoding appearing in (3.2). Since only one instance of any rule in the LSTS can be used, we would only be allowed to encode runs that use an action of M once. In order to overcome this problem, here, instead of using propositional rules, we use $6(n+2)$ first-order actions of the form:

$$\begin{array}{ll}
S_{i,q}(t)R_{i,\xi} \to_A \exists t_n.F_{i,\gamma}(t_n)R_{i,\xi}, & F_{i,\gamma}(t)R_{i,\xi} \to_A F_{i,\gamma}(t)H_{i,\gamma}(t),\\
F_{i,\gamma}(t)H_{i,\gamma}(t) \to_A G_{i,\gamma}(t)H_{i,\gamma}(t), & G_{i,\gamma}(t)H_{i,\gamma}(t) \to_A G_{i,\gamma}(t)R_{i,\eta},\\
G_{i,\gamma}(t)R_{i,\eta} \to_A S_{i_D,q'}(t)R_{i,\eta}, & S_{i,q}(t) \to_A S_{i,q}.
\end{array} \tag{3.3}$$

where $i=0,1,..,n+1$. The initial state contains a variable $S_{i,q}(c)$ with some constant c and the goal state is the accepting configuration with a propositional variable $S_{j,q}$ (of arity zero). Intuitively, the first five rules above are used in the same way as before to encode M's actions of the form $S_{i,q}R_{i,\xi} \to_A S_{i_D,q'}R_{i,\eta}$, but, now, we create a new constant, t_n, every time we apply the first rule. This allows us to encode runs where the same action of M is used more than once, since, for each use of this action, we use a different instance of the rules in (3.3). Moreover, since in the accepting state one is not interested in the constant t appearing in the variables $S_{i,q}(t)$, we use the last rule in (3.3) when the accepting state is reached. Notice that after this last action is used, no other rule in (3.3) is applicable. $\square$

However, regarding the upper bound for the case when systems are progressing, it is no longer possible to use the same algorithm use in the proof of Theorem 3.5. The problem is that one needs to check that the plan is progressing, which requires to check that in the whole plan, any instance of any action is used at most one. In order to do so, it seems that one needs to remember the whole plan. It remains to be seen if the restriction of progressing LSTS to only monadic actions gives lower complexity result.

Theorem 3.11. *The problems of progressing weak plan compliance, progressing plan compliance, and the secrecy problem for progressing plans for LSTSes with balanced actions that can create nonces are in EXPSPACE.*

Proof The algorithms are similar to algorithms for the plans that are not necessarily progressing but there are additional steps for checking of the progress, both when checking for compliance and when constructing a plan. Each time an action is nondeterministically chosen as the candidate for the next action in the plan, the algorithm has to check that it has not already been used. It does that by comparing it to the actions used so far. Therefore we have to store the plan. As we have already seen an action takes $O(|W|)$ space, but the length of a plan can be up to $L_T(m)$ which is exponential in respect to the given inputs. □

3.4 Complexity in Not Necessarily Balanced Systems with Nonces

In this section we show that the weak plan compliance problem for progressing systems with possibly unbalanced actions that can create nonces as well as system compliance problem for systems with possibly unbalanced actions with nonces are undecidable. This is shown by faithfully encoding the two counter Minsky Machine in a similar way as in [26] and by encoding the Turing machine with unbounded tape similarly to [18], respectively.

3.4.1 Undecidability in Progressing Systems

We use a standard two-counter Minsky machine M of a certain form as shown in Figure 3.1. We assume the instructions alternate between instructions for register 1 and instructions for register 2. Instructions labeled by a_i will be run by Alice, instructions labeled by b_j will be run by Bob.

$$\begin{array}{ll}
\textbf{Jump} & a_i : \textbf{goto } b_j; \\
\textbf{Add} & a_i : r_1 := r_1 + 1; \textbf{goto } b_j; \\
\textbf{Substract} & a_i : r_1 := r_1 - 1; \textbf{goto } b_j; \\
\textbf{0-test} & a_i : \textbf{if } (r_1 = 0) \textbf{ goto } b_j \textbf{ else goto } b_k; \\
\\
\textbf{Jump} & b_j : \textbf{goto } a_k; \\
\textbf{Add} & b_j : r_2 := r_2 + 1; \textbf{goto } a_k; \\
\textbf{Substract} & b_j : r_2 := r_2 - 1; \textbf{goto } a_k; \\
\textbf{0-test} & b_j : \textbf{if } (r_2 = 0) \textbf{ goto } a_k \textbf{ else goto } a_l;
\end{array} \tag{3.4}$$

Figure 3.1: The form of Minsky machine instructions

Each instruction is labeled with the unique label. States a_1 and a_0 are the initial and final states of M, respectively. Furthermore, a_0 is a halting state so it is distinct from the label of any of M's instructions. M's configuration where M is in state m, and k_1 and k_2 are the current values of counters r_1 and r_2, respectively, is denoted by $(m; k_1, k_2)$. A computation performed by M is a sequence of M's configurations such that each step is made by one of the above instructions:

$$(a_1; n, 0) \xrightarrow{a_1} \cdots \longrightarrow (a_i; k_1, k_2) \xrightarrow{a_i} (b_j; k_1', k_2') \xrightarrow{b_j} \cdots$$

A *terminating computation* is one that ends in a configuration $(a_0; *, *)$, that is, the final state a_0 with any values in the counters.

Following [26] and [23] we embed a given Minsky machine M into a local state transition system. The soundness and completeness of the translation imply the undecidability of the problem.

Theorem 3.12. *The problems of progressing weak plan compliance and progressing plan compliance, as well as the secrecy problem for progressing plans are undecidable for LSTSes that may have unbalanced actions that can create nonces.*

Proof (Sketch) We modify the encoding in [26] by introducing a dummy one arity predicate $Aux(\cdot)$ to all the actions in their specification. This predicate is used to allow one to circumvent the progressing condition with the use of nonces. By using this predicate, one is able to repeat the effect of an instruction of the machine without needing to repeat an instance of an action twice. Bob's actions are shown below:

$$\begin{array}{lll}
Aux(x)\ S_j^B & \rightarrow_B & \exists y.Aux(y)\ R^B\ C_1(a_k) \\
Aux(x)\ S_j^B & \rightarrow_B & \exists y.Aux(y)\ R^B\ R_2\ C_1(a_k) \\
Aux(x)\ S_j^B\ R_2 & \rightarrow_B & \exists y.Aux(y)\ R^B\ C_1(a_k) \\
Aux(x)\ S_j^B & \rightarrow_B & \exists y.Aux(y)\ R^B\ C_1(a_{\tilde{k}}) \\
Aux(x)\ S_j^B\ R_2 & \rightarrow_B & \exists y.Aux(y)\ R^B\ R_2\ C_1(a_l) \\
Aux(x)\ R^B\ C_2(b_j) & \rightarrow_B & \exists y.Aux(y)\ S_j^B
\end{array}$$

The initial state contains exactly one fact of the form $Aux(t)$, where t is some constant. All the actions above remove such a fact and include fact $Aux(y)$ with a new constant y. This forces that all instances of actions used in a plan to be different from each other. Moreover, as in [26], to avoid the unintended non-determinism in the above actions specifying the command **if** $(r_2 = 0)$ **then** a_k **else** a_l, we make of critical states. In particular, states containing either the facts $C_1(\hat{a_j}), R_1$ or the facts $C_1(\tilde{b_j}), R_2$ are critical. □

Remark We would like to point out that this work on progressing systems is a preliminary work and express our doubts on this approach to progressing. Although we were able to get an improvement in the complexity of the problems when only a bounded number of nonce creations was allowed, in other cases our notion of progressing backfired, see Theorem 3.3. It turned out that in the systems with unbounded number of nonce creations progressing was too expensive. We found progressing with nonces quite complicated and unclear. We are currently working on an alternative notion of progressing which would provide more efficient systems.

3.4.2 Undecidability of System Compliance

For LSTSes with possibly unbalanced actions that cannot create fresh values, it was shown in [26] 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 further allow actions to create fresh values. We show next that when actions are possibly unbalanced and can create fresh values, it turns out that the system compliance problem is also undecidable.

Theorem 3.13. *The system compliance problem for general LSTSes with actions that can create fresh values is undecidable.*

Proof The proof relies on undecidability of acceptance of Turing machines with unbounded tape. The proof is similar to the undecidability proof of multiset rewrite rules with existentials in [18].

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 $x_1x_2\dots x_n$, is written in cells 1, 2,..., n on the tape. In addition, a special marker # is written in the $(n{+}1)$-th cell.

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

(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 signature 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 $a_0, a_1, \ldots, a_m : \mathit{symbol}$ to represent symbols of the tape alphabet with a_0 denoting blank; constants $q_0, q_1, \ldots, q_f : \mathit{state}$ for the machine states, where q_0 is the initial state and q_f is the accepting state; and finally constants $\$, c_1, \ldots, c_n, \# : \mathit{cell}$ 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 $\mathit{Curr} : \mathit{state} \times \mathit{cell}$, $\mathit{Cont} : \mathit{cell} \times \mathit{symbol}$ and $\mathit{Adj} : \mathit{cell} \times \mathit{cell}$ 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:

$$\mathit{Adj}(c, \#) \to \exists c'. \mathit{Adj}(c, c')\ \mathit{Adj}(c', \#)\ \mathit{Cont}(c', \#)\,. \tag{3.5}$$

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:

$$\mathit{Curr}(q_i, c)\ \mathit{Cont}(c, a_s)\ \mathit{Adj}(c', c) \to \mathit{Curr}(q_j, c')\ \mathit{Cont}(c, a_t)\ \mathit{Adj}(c', c). \tag{3.6}$$

Notice that we move to the left by using the fact $\mathit{Adj}(c', c)$ 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:

$$\mathit{Curr}(q_i, c)\ \mathit{Cont}(c, a_s)\ \mathit{Adj}(c, c') \to \mathit{Curr}(q_j, c')\ \mathit{Cont}(c, a_t)\ \mathit{Adj}(c, c')\,. \tag{3.7}$$

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 tape maintenence rule 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 $\$x_1 x_2 \ldots x_k\#$ is written left-justified on the otherwise blank tape, we represent it by the following initial configuration of $T_{\mathcal{M}}$

$$\begin{gathered}\mathit{Cont}(c_0, \$)\ \mathit{Cont}(c_1, x_1) \ldots\ \mathit{Cont}(c_k, x_k)\ \mathit{Cont}(c_{k+1}, \#) \\ \mathit{Curr}(q, c)\ \mathit{Adj}(c_0, c_1)\ \ldots, \mathit{Adj}(c_k, c_{k+1})\,.\end{gathered} \tag{3.8}$$

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 (3.5), (3.6) and (3.7), which enforce that any reachable configuration has exactly one occurrence of $Curr(q, c)$. Moreover, any reachable configuration is of the form similar to (3.8), 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. □

Table 3.2 gives the summary of complexity results and the comparison between the progressing and not necessarily progressing problems.

Table 3.2: Summary of the complexity results for the weak plan compliance and the secrecy problems. New results appearing in this paper are marked with a $\star$.

Weak plan compliance problems		Progressing	Not necessarily progressing
Balanced Actions	Bounded N^o of Nonces	NP-complete*	PSPACE-complete [27]
	Unbounded N^o of Nonces	PSPACE- hard* in EXPSPACE*	PSPACE-complete*
Actions not necessarily balanced		Undecidable*	Undecidable [26]

Secrecy problems		Progressing	Not necessarily progressing
Balanced Actions	Bounded N^o of Nonces	NP-complete*	PSPACE-complete [27]
	Unbounded N^o of Nonces	PSPACE-hard* in EXPSPACE*	PSPACE-complete*
Actions not necessarily balanced		Undecidable*	Undecidable [18]

Chapter 4

Balanced Protocol and Intruder Theories

In this section we investigate whether malicious agents, or intruders, with the same capabilities of the other agents are able to discover some secret information. In particular, we assume that all agents have a bounded storage capacity, that is, they can only remember, at any moment, a bounded number of facts. This is technically imposed by considering LSTSes with only balanced actions. Namely, if we restrict actions to be balanced, they neither increase nor decrease the number of facts in the system configuration, therefore throughout a run the size of the configurations 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 all the agents in the system, including the inside adversary.

4.1 Representing Protocols with Multiset Rewriting Rules

Protocols and the relevant security problems can be modeled with multiset rewriting rules, see [18]. In that scenario a set of rewrite rules, or a theory, was proposed for modeling the standard Dolev-Yao intruder. Here, we adapt that theory to model instead an intruder that is an agent of the system, *i.e.* adversary is an insider not an outside attacker. Furthermore we model 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 nonces.

Generally, actions in LSTS can increase or decrease the number of facts in the global configuration. Balanced actions, however, do not change the number of facts in al configuration. A total size of the configuration of a balanced system is constant. There is a fixed number of number of facts in a configuration, say m facts. Also, each fact is of a bounded size, i.e. there are at most a slots for the term-symbols, resulting in a total of $m \cdot a$ slots in a configuration. Balanced actions are therefore useful in modeling systems with fixed resources. Intuitively, a global configuration is a fixed amount of total system memory. Each agent has a buffer or database of a fixed size and there's the remaining fixed public buffer. During the collaboration with other agents values in the relevant buffers are updated or erased to leave empty fields.

In reality, balanced systems are more flexible than that. There is one global buffer of a fixed size, and the agents are free to release some fields for use by the group and claim others from the group. The model allows for a potential fight for space resources which could result in a form of denial of service.

While at first balanced systems might seem very restrictive compared to general systems which allow unbalanced actions, in practice we are still able to model most scenarios in a natural way using balanced systems. It is possible to transform any unbalanced system into a balanced one. Assume that our language has a special constant $*$. A predicate of the form $P(*)$ can be interpreted as saying that the field P is empty. We can add $P(*)$ facts to the pre-condition or alternatively to the post-condition of an unbalanced action so that it becomes balanced. This modification can affect reachability. Some configurations become no longer reachable. For example, if along a plan of an unbalanced LSTS a configuration contains more than m facts, the goal configuration is no longer reachable in the modified system. Therefore that particular plan is not a plan of the balanced system. However, this is not a really significant problem, it is, as we have already called it, a fight for space. Although using only balanced actions forces us to fix the number of fields of this database, there is no a priori bound on the number of fields we may choose.

We will be interested in the same secrecy problem as in [18], namely, in determining whether or not there is a plan which the intruder can use to discover a secret. We assume that in the initial state some agent has a secret s, that is, an agent A owns a fact $F_A(s')$ with the secret s being a subterm of s'. Intruder discovers the secret once he owns a fact containg the secret term s. It's been shown in [18] that the secrecy problem is undecidable. However, since in modeling a bounded memory systems we only allow balanced actions, the secrecy problem becomes PSPACE-complete.

4.1.1 Protocols and Intruders with Bounded Memory

Since we need to take care of the limited memory available to the agents, we relax the form of the protocols introduced in [18]. We modify the well-founded protocol theory and the two-phase intruder theory introduced in [18] so that all the rules become balanced.

Instead of well-founded protocol theories, we introduce semi-founded protocol theories, where protocol roles are not necessarily created before any protocol instance starts, but can be created while other protocol instances are already running. That allows us to model an unbounded number of protocol sessions. Even though we are able to represent an unbounded number of protocol sessions in a fixed (bounded) memory, we are clearly unable to represent a concurrent run of any number of protocols, such as the generalization of the necessarily parallel attack on an artificial protocol in [31].

Having only a bounded amount of memory, the intruder may have to manage its memory capacity in order to discover a secret. Whenever the intruder needs to create a fresh value or remember some public fact, he will have to check whether there is some free fact available. We also adapt the two-phase intruder theory from [18] so that our memory bounded Dolev-Yao intruder uses his memory economically, as wisely as possible. This includes deleting data and digesting only those messages and parts of messages that contain useful data.

Furthermore, we extend the basic signature given in [18] that was related mainly to the Needham-Schroeder protocol, to be able to represent other protocols and the related anomalies as well. For example, our model will include symmetric encryption, encryption with composed keys, key generation, messages with signatures and timestamps.

Notice that since the size of configurations is bounded, there is a bound on the number of messages in the public domain. Although the intruder is one of the agents and can therefore use all public facts (provided he has the necessary rules), for our formal results it will be useful to distinguish the memory storage capacity of the intruder from the storage capacity of all the other agents of the system. We use the fact $R(*)$ private to the intruder to specify a memory slot available only to the intruder, while the fact $P(*)$ is public and can be used by the remaining agents and by the network. Notice that technically the intruder also has access to $P(*)$ fact. However, when we specify his actions, we will make sure that he only creates memory facts, *i.e.*, stores data, whenever he has a free memory slot $R(*)$. He might however free a $P(*)$ when, for example, he intercepts a message from the network. As we show in the next sections, the intruder might have to manage his memory capacity in order to discover a secret. Whenever the intruder needs to create a fresh value or remember some public fact, he will have to check whether there is some empty fact available.

In our analysis of the memory needed by the intruder to carry out an anomaly or to compose messages, we will often provide bounds with respect to the number of facts that are public or that belong to agents different from the intruder. For this purpose, we introduce the notion of size of configurations modulo the intruder.

Definition 4.1. The *size modulo the intruder of a configuration* $\mathcal{S}$ is the number of public facts in $\mathcal{S}$ plus the number of private facts in $\mathcal{S}$ that do not belong to the intruder.

When we introduce the theory of our memory bounded intruder with balanced actions and protocol theories also with balanced actions, we will make sure that the size modulo the intruder of configurations in a plan derived from these theories is always the same.

In the next sections, we introduce the LSTS with balanced actions that includes a balanced protocol theory, introduced in Section 4.2, a balanced two-phase intruder theory with a memory management theory, both introduced in Section 4.3. We also illustrate in Section 4.2.1, a balanced protocol theory for the Needham-Schroeder public key exchange protocol [33].

4.2 Balanced Protocol Theories

We modify the rules from [18] that specify protocol theories so that only balanced actions are used. We also relax the protocol form imposed in [18], 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.

Since whenever a protocol role is created in [18] an extra fact is added to the configuration, when using balanced systems one is not allowed to generate an unbounded number of protocol roles. The number of roles in balanced systems with well-founded theories is bounded by the number of empty facts available after the initialization phase. Imposing bounds on the number of protocol session is not always acceptable. In reality it is often expected that agents establish secure channels using the same protocols an unbounded number of times and creating an unbounded number of nonces. For instance, in online banking, a bank customer checks his online statement, accessing his personal online bank homepage and inserting his online PIN number, possibly an unbounded number times.

To overcome this problem, we introduce semi-founded protocol theories, where protocol roles are not necessarily created before any protocol instance starts, but can be created while other protocol instances are already running. Hence, whenever a protocol instance finishes, agents can delete them from their memory and start new protocol instances, allowing an unbounded number of protocol runs.

Before we enter into the details of semi-founded protocol theories, we introduce some more notation involving encryption taken from [18].

We introduce the signature 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 encryption keys, *dkey* for decryption keys, and a sort *msg* for any type of message. Here we use order-sorted signature and have *msg* as a super-sort because different types of data can be encrypted and in that encrypted form their sorts are unknown, until the message is decrypted. The following order relations hold among these sorts:

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

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

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

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 [18].

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 [18], 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*.

Predicates used in the protocol theory will depend of the particular protocol that is represented. For instance, all the predicates of the well-known Needham-Schroeder protocol [33], discussed in the next section, are shown in Figure 4.1. 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
cipher :	cipher text (encrypted)
nonce :	nonces
msg :	data of any type

Subsorts :
$nonce < msg, \quad cipher < msg,$
$ekey < msg, \quad dkey < msg$

Functions :	
$enc : ekey \times msg \to cipher$:	encryption
$\langle \cdot, \cdot \rangle : msg \times msg \to msg$:	pairing

Predicates :	
$GoodGuy(ekey, dkey)$:	identity of an honest participant
$BadKey(ekey, dkey)$:	keys of a dishonest participant
$KP(ekey, dkey)$:	encryption key pair
$AnnK(ekey)$:	published public key
$A_0(ekey)$:	Role state 0 for initiator
$A_1(ekey, ekey, nonce)$:	Role state 1 for initiator
$A_2(ekey, ekey, nonce, nonce)$:	Role state 2 for initiator
$B_0(ekey)$:	Role state 0 for responder
$B_1(ekey, ekey, nonce, nonce)$:	Role state 1 for responder
$B_2(ekey, ekey, nonce, nonce)$:	Role state 2 for responder
$N_S(cipher)$:	(i = 1, 2, 3) encrypted message (sent)
$N_R(cipher)$:	(i = 1, 2, 3) encrypted message (received)
$N(cipher)$:	encrypted message on the network (sent or received)

Figure 4.1: Signature for the Needham-Schroeder Public Key Protocol.

Predicate $KP(ekey, dkey)$ represents the pair of associated encryption and decryption (public and private) keys. Predicate $AnnK(ekey)$ indicates that a public key has been published. The distinction between honest participants and participants with compromised keys is achieved with two predicates, $GoodGuy(ekey, dkey)$ and $BadKey(ekey, dkey)$. There are also role state predicates for the initiator $A_0(ekey)$, $A_1(ekey, ekey, nonce)$, $A_2(ekey, ekey, nonce, nonce)$, role state predicates for the responder $B_0(ekey)$, $B_1(ekey, ekey, nonce, nonce)$, $B_2(ekey, ekey, nonce, nonce)$ and the network predicates $N(cipher)$, $N_{Ri}(cipher)$ and $N_{Si}(cipher)$. We will further explain the intended interpretation and use for the predicates when we get to the examples.

We now introduce the necessary definitions.

Definition 4.2. 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 request on excluding empty facts in preserving and creating facts captures the idea that the empty facts hold no information.

Definition 4.3. 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 $\mathcal{S}$.

Definition 4.4. 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, such that for each rule $L \to R$ in $\mathcal{A}$ is balanced and there is exactly one occurrence of a state predicate in L, say S_i, and exactly one occurrence of a state predicate in R, say S_j, such that $i < j$. We call the first role state, S_0, *initial role state*, and the last role state S_k *final role state*. Only rules with final role states can have an empty fact in the post-condition.

Defining roles in this way, ensures that each application of a rule in $\mathcal{A}$ advances the state forward. Each instance of a role can only result in a finite number of steps in the derivation. The request on empty facts formalizes the fact that one of the participants, either the initiator or the responder, sends the “last” protocol message. For example, with the Needham-Schroeder Public Key protocol the responder goes to the final state and does not send any message at that point. With the Kerberos it could be either the responder or the initiator, depending on whether acknowledgement is required or not.

The following definition formalizes our previous intuition that roles can be created not only before any protocol instance has started, but also while protocol instances are running. In particular, we add a rule that allows one to delete a role state which generates an empty fact. This empty fact can then be used to generate a new role state, starting hence a new protocol instance.

Definition 4.5. If $\mathcal{A}_1, \dots, \mathcal{A}_k$ 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 $Q_1(\vec{x}_1) \dots Q_n(\vec{x}_n)$ is a finite list of persistent facts not involving any role states, and S_0 is the initial role state for one of theories $\mathcal{A}_1, \dots, \mathcal{A}_k$, or the form

$$S_k \to P(*)$$

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

The following definition relaxes well-founded protocols theories in [18] in order to accommodate the creation of roles while protocols are running.

Definition 4.6. A pair $(\mathcal{P}, \mathcal{I})$ is a *semi-founded protocol theory* if $\mathcal{I}$ is a finite set of persistent facts (called *initial set*), and $\mathcal{P} = \mathcal{R} \uplus \mathcal{A}_1 \uplus \cdots \uplus \mathcal{A}_n$ where $\mathcal{R}$ is a role regeneration theory involving only facts from $\mathcal{I}$ and the initial and final roles states of $\mathcal{A}_1, \dots, \mathcal{A}_n$, and $\mathcal{A}_1, \dots, \mathcal{A}_n$ are balanced role theories. For role theories $\mathcal{A}_i$ and $\mathcal{A}_j$, with $i \neq j$, no role state predicate that occurs in $\mathcal{A}_i$ can occur in $\mathcal{A}_j$.

The finite initial set of persistent facts normally contains all the information necessary to start protocol sessions, for instance, shared and private keys as well as the names of the participants of the network. Intuitively, this means that we analyze protocols after all initialization, such as key distribution, has been already completed.

Remark. We could hide the complexity of our problems by encoding complex problems into initialization theory, but that would not result in real complexity of the protocol analysis. To avoid this we allow initialization theories to consist only of a finite number of ground facts and no rewrite rules. Intuitively, this means that we analyze decidability and complexity of the role generation and protocol execution phases, under the assumption that initialization has already been completed.

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. While Figure 4.1 shows types and predicates needed for modeling Needham-Schroeder protocol, Figures 4.2, 4.3 and 4.4 show the extended typed signature.

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 :

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

Functions :

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

Figure 4.2: 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 4.3: Predicates for the Protocol theories

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 [14]. Such attacks are prevented by typed signatures 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 protocols that use digital signatures. We represent signatures 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, $TGSKey$ 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.

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 4.4: Predicates specific to the Kerberos Protocols

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 cerificates 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.

4.2.1 Semi-founded Protocol Theory for the Needham-Schroeder Protocol

This section illustrates semi-founded protocol theory defined in the previous section by using the Needham-Schroeder for public key exchange protocol [33]. In particular, facts $P(*)$ represent empty facts available to all participants. We use predicates depicted in Figure 4.1. Figure 4.5 presents the protocol using a common informal notation. The complete Needham-Schroeder Public Key protocol includes distribution of public keys by a trusted server, those steps are omitted here.

$$\begin{array}{l}A \longrightarrow B: \ \{A, n_a\}_{kb}\\ B \longrightarrow A: \ \{n_a, n_b\}_{ka}\\ A \longrightarrow B: \ \{n_b\}_{kb}\end{array}$$

Figure 4.5: Needham-Schroeder Public Key Protocol.

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), Bob replies with a nonce of his own, n_b, along with initiator's (Alice's) nonce, encrypted with Alice's public key. Alice receives the message encrypted with her public key, decrypts it, and if it contains her nonce, Alice replies by returning Bob's nonce, encrypted with his public key. At the end they believe that they are communicating with each other.

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

Figure 4.6: Semi-founded protocol theory for the Needham-Schroeder Protocol.

The rules of a semi-founded Needham-Schroeder Protocol theory are shown in Figure 4.6. Predicate $KP(ekey, dkey)$ denotes the pair of associated encryption and decryption (public and private) keys. Predicate $AnnK(ekey)$ indicates that a public key has been published. The distinction between honest participants and participants with compromised keys is achieved by using respectively the following two predicates *GoodGuy* and *BadKey*. There are also role state predicates A_i and B_i, where $0 \leq i \leq 2$, and a network predicate N. For simplicity, 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).

The protocol execution is separated into stages. There is an initialization phase, represented by the initial set of facts, that distributes key pairs to participants and announces their public keys which is essential for the protocol communication. Following the initialization phase, there is a protocol execution phase. With the rules from the role regeneration theory each agent, within their bounded memory, may choose to carry out the protocol some number of times, in any combination of roles. For example a principal A may play the role of initiator twice, and responder once, at the beginning of a protocol run. Then the rules characterizing different roles are applied. Later on, A may decide to replay some of the roles or choose different ones.

Initial set of facts The initial set of facts includes persistent facts $KP(k_e, k_d)$ denoting pairs of public and private (encryption and decryption) keys. The $GoodGuy(k_e, k_d)$ facts represent honest principals participating in the protocol exchange, parameterized by their public and private keys. The $BadKey(k_e, k_d)$ facts specify a number of compromised key pairs. Such keys appear to belong to valid principals, but their private keys are known to the intruder who can then play the role (or a part of the role) of an honest principal if he wants. Therefore there is no need to include both $GoodGuy(k, k')$ and $BadKey(k, k')$ facts for the same keys. Key distribution is accomplished by all principals announcing their public keys through $AnnK(k_e)$ facts indicating a public key that is available for communication, so from this point the honest participants can not distinguish the good guys from the bad guys. Notice that the number of participants is not unlimited since the size of the configurations is bounded. One way to overcome this problem is by allowing the deletion of key pairs from system configurations. That would, however, disallow some principals from participating further in the protocol exchange.

Role regeneration theory In the role regeneration theory facts representing initial role states are created and facts representing final role states are deleted. Predicates $A_0, A_1, A_2, B_0, B_1, B_2$ are role state predicates. Rules ROLA and ROLB generate the roles for any principal to act in the role of either Alice (the initiator) or Bob (the responder). A_0 and B_0 denote the initial role states for the A and B roles, respectively, parameterized by the public key (principal) acting in that role. The rules ERASEA and ERASEB allow an unbounded number of roles to be created: Deleting a role creates an empty fact that can be reused by ROLA and ROLB rules to create a new protocol instance. Notice that the last two rules were not necessary in [18] since agents were allowed to have unbounded memory. Here on the other hand, an agent needs to allocate a piece of his memory in order to create a new session.

Protocol Role Theories Rules from the protocol theories replace initial role state facts with facts representing other role states. This replacement always replaces a role state, A_i, by a role state, A_j, with a greater number associated to it, that is $j > i$. The protocol role theories in Figure 4.6 are derived from the specification of the Needham-Schroeder protocol in Figure 4.5. Theory $\mathcal{A}$ corresponds to the role of Alice, and theory $\mathcal{B}$ corresponds to Bob. Predicate N represents network messages being sent and received. With no intruder present we model the protocol execution by having a single predicate N for network messages. However, in the rest of this paper, as in [18], protocol execution assumes that the intruder acts as the network, so we use predicates N_S and N_R. This will be explained in more detail later.

In rule A1, which corresponds to the first line of the protocol, a principal k_e, in its initial state A_0, decides to communicate to another principal k_e', whose key has been announced. A new nonce x is generated, along with a network message corresponding to the first message sent in the protocol, and Alice remembers the values of x and k_e' and moves to the next state A_1. Rule B1 corresponds to the second step of the protocol. Principal k_e, in the initial state B_0, responds to a message from the network if it is of the expected format (*i.e.*, encrypted with k_e's public key, and with the identity of a participant whose key has been announced, embedded inside). Then he generates another nonce, replies to the message, sending both nonces, and moves to the next state B_1 remembering all the information (the two nonces and the two principals). Similarly, rule A2 corresponds to the third line of the protocol, and rule B2 corresponds to the implicit step where the responder actually receives the final message.

4.3 Memory Bounded Intruder Theory

We now adapt the two-phase intruder theory from [18] in such a way that all the rules in the resulting theory are balanced and represent an intruder with bounded memory. In order to do so, we use empty facts $R(*)$ to fill the left or the right side of rules to make them balanced.

As the Dolev-Yao intruder specified in [18], our memory bounded intruder is still able, provided he has enough memory slots available, 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.

With memory bounds in mind, we introduce memory maintenance theories that delete facts, *i.e.*replace memory facts with empty facts.

Definition 4.7. 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.

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.

The balanced two-phase intruder theory with the rules similar to those in [18] plus the additional rules for new sorts and types of encryption is depicted in Figure 4.7. Additional rules that enable the intruder to use his memory more cleverly are depicted in Figure 4.9. Finally, his memory maintenance theory is depicted in Figure 4.8.

	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 4.7: Two-phase Intruder theory.

Memory maintenance rules:

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

Figure 4.8: Memory maintenance theory.

Decomposition Rules:

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

Composition Rules:

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

Memory maintenance rules:

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

Figure 4.9: Additional rules for the Two-phase Intruder theory.

Rules REC and SND, as depicted in Figure 4.7, connect the intruder to the network used by the participants for protocol exchange. The REC rule intercepts a message from the network and saves it as a decomposable fact. The SND rule sends composed messages onto the network. As in [18], the protocol execution assumes that the intruder acts as the network, *i.e.*, all messages sent by any agent are received by the intruder and all messages received by any agent are sent by the intruder. In our formalization, this corresponds to transforming the sent messages, represented by the facts $N_S(m)$, into messages that can be received by an honest participant, represented by the facts $N_R(m)$. With no intruder(network) protocol cannot execute.

Intruder uses the COMP rule to compose parts of messages into pairs, while he uses the DCMP rule for decomposition of pair terms into smaller ones. 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. The ENC and DEC rules allow the intruder to decrypt a message if it knows the private key, and to generate encrypted message from known public keys. Rules LRNA and DECA are decomposition rules with auxiliary facts that handle the case when the message can't be decrypted because the private key isn't currently known to the intruder. LRNA remembers the encrypted message with the special "Auxiliary" predicate, A. The DECA rule allows Auxiliary messages to be decrypted at a later time, if the decryption key becomes known. 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. GEN rule allows the intruder to generate new facts (*i.e.*, nonces) when needed. GENM rule generates a generic message to perform "ticket anomaly" in Kerberos 5 shown in Section 5.6.1. 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.

Notice the role of the facts $P(*)$ and $R(*)$ in the rules. For instance, in the REC rule, when the intruder intercepts a message from the network, one of the intruder's free memory slots, $R(*)$, is replaced with a fact containing the intercepted message and a free memory slot, $P(*)$, belonging to the other agents replaces the network fact. The intruder is not allowed to intercept a new network fact if he does not have any free memory slot left. Similarly, in the GEN rule, the intruder needs a fact $R(*)$ in order to generate a nonce. 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 4.8. Using these rules intruder can forget any facts stored in his memory which are of the form $M_?$. This contrast with [18], where these predicates were persistent throughout a run, that is, they were always present in the intruder's memory. Since in [18] 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. Therefore we extend intruder's theory with additional rules depicted in Figure 4.9 that enable the intruder to use his memory more economically.

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 4.7 to include this rule. Alternatively, in a standard trace we could simulate this rule with the following derivation:

$$\begin{aligned} &N_S(m)\ R(*)\ \to_{REC}\ D(m)\ R(*)\ \to_{DM}\ M_s(m)\ R(*)\ \to_{USES} \\ &\quad M_s(m)\ C(m)\ \to_{SND}\ M_s(m)\ N_R(m)\ \to_{DELMS}\ N_R(m)\ R(*) \end{aligned}$$

The intruder's 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 4.2 to consider the empty facts $B(*)$ as well.

4.3.1 Two-phase Intruder Theory and Memory Maintenance Theory

As in [18] intruder's theory is a two-phase theory. In order to formalize this intuition, we follow [18] and define the following weighting function.

Definition 4.8. A *weighting function* is a function $\omega : \mathcal{F} \to \mathbb{N}$ that maps atomic formulas to numeric weights. We use $\omega(A)$ to denote the weight of the atomic formula A. The relative weight of formulas must be preserved under substitution, *i.e.*, if A and B are atomic formulas and σ is a substitution, then

$$\omega(A) > \omega(B) \Rightarrow \omega(\sigma A) > \omega(\sigma B).$$

We define the weight of empty facts to be zero, that is:

$$val(R(*)) = val(P(*)) = val(B(*)) = 0\,.$$

We use weighting functions to guarantee termination of normalized derivations. Many weighting functions could be used. Here, we use such functions that calculate the weight of facts based on the predicate symbol used and the size of the fact. We define a strict (non-reflexive) partial-ordering on the predicates of a theory. A particular theory has a particular ordering. For instance, for the intruder theory specified in Figure 4.7 we could use the following weighting function:

$$\omega 1(F(x)) := 10 \cdot (|F(x)| - 1) + val(F)$$

where

$$val := \{(N,4), (N_S,4), (N_R,4), (D,3), (A,2), (M_?,1), (C,3), (R,0), (P,0), (B,0)\},$$

and the value "10" was arbitrarily chosen to be larger than any of the values appearing in the val function.

Definition 4.9. A rule $r = L \to R$ is a *decomposition rule* with respect to weighting function ω if the total weight of the facts in R is less than the total weight of the facts in L. A rule $r = L \to R$ is a *composition rule* with respect to weighting function ω if the total weight of the facts in R is greater than the total weight of the facts in L. By total weight of a multiset of facts we mean the sum of the weights of all the facts in the multiset.

For example, $D(\langle x,y\rangle)\ R(*) \to D(x)\ D(y)$ is a decomposition rule with respect to weighting function $\omega 1$, and $C(x)\ C(y) \to C(\langle x,y\rangle)\ R(*)$ is a composition rule with respect to weighting function $\omega 1$.

As in [18] intruder's theory $\mathcal{M} = \mathcal{C} \uplus \mathcal{D}$ is a two-phase theory. Such a theory provides normalized derivations, *i.e.*, derivations where all rules from the decomposition theory $\mathcal{D}$ are applied before any rules from the composition theory $\mathcal{C}$.

Definition 4.10. A theory $\mathcal{M}$ is a *two-phase theory* if its rules can be divided into two theories, $\mathcal{M} = \mathcal{D} \uplus \mathcal{C}$, where $\mathcal{C}$ contains only composition rules, $\mathcal{D}$ contains only decomposition rules, and no rules in $\mathcal{C}$ precede any rules in $\mathcal{D}$.

Rules from the intruder theory depicted in Figure 4.7 and Figure 4.9 can be divided into composition and decomposition rules. Rule REC is a decomposition rule and rule SND is a composition rule. Therefore, as per Definition 4.10, this is a two-phase intruder theory with respect to weighting function $\omega 1$.

In order to have a two-phase intruder theory we need to treat empty facts with care, *i.e.*, we do not say that empty facts are created nor consumed as per Definition 4.2. Empty facts generated by some composition rules such as COMP, appear in the preconditions of some decomposition rules. Therefore the theory depicted in Figure 4.7 is indeed a two-phase intruder theory as per Definition 4.10.

Definition 4.11. A *normalized derivation* is a derivation where the rules from the decomposition theory are applied eagerly until no more decomposition rules are enabled in the configuration and only then rules from the composition theory are applied.
An *extended normalized derivation* is a normalized derivation in which all REC rules are applied at the beginning of the derivation, and all SND rules are applied at the end of the derivation.

The intuition behind the definition of extended normalized derivations is that intruder first intercepts messeges, then decomposes them and sends intended messages only after he has composed all of them.

The following lemma states that any derivation obtained using a two-phased intruder can be transformed into a normalized derivation provided the intruder has enough memory. In other words, if the intruder has enough memory, one only needs to search for normalized derivations.

Lemma 4.12. In a two-phase theory all derivations can be transformed into the (extended) normal form, provided its first configuration has kl empty facts, where l is the size modulo the intruder of configurations and k is the upper bound on the size of facts.

Proof Let $\mathcal{M} = \mathcal{D} \uplus \mathcal{C}$. Since no rules from $\mathcal{C}$ enables rules in $\mathcal{D}$, all rules from $\mathcal{D}$ can be applied before any rules from $\mathcal{C}$. We have to consider memory requirements for such rearrangement of rules. We look at the rules that free memory slots: COMP and

SND. Moving them forward in a derivation could result in not having enough empty facts to enable other rules earlier in a derivation. Along the given derivation there could be a configuration with no empty facts followed by the COMP rule. COMP rule frees one slot so another rule that consumes an empty fact can be applied, for example:

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

When we try to switch DCMP and COMP rules, we cannot do that because there is 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:

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

Note that now there is an empty fact in the resulting configuration, but there are no empty facts in the configuration which enables the COMP rule. Therefore, we need at most one additional empty slot for permuting a COMP rule in the derivation. We need to see how many of such permutations could there be. Notice that there is a bounded number of decomposition rules in a derivation, since only intercepted messages are decomposed and only l messages of size k could be intercepted. Therefore, there could be at most $(l-1)(k-1)$ decomposition rules in a derivation. On the other hand, intruder can compose messages (create C facts) within his memory capacity, so the number of composition rules in a derivation is not bounded by l, but with the size of intruder's memory instead.

With above in mind, we transform the derivation into a normalized derivation by moving all decomposition rules to the beginning of the derivation, starting from the leftmost decomposition rule, d, that needs to be moved. As we have shown before, with respect to memory, the problematic configurations are ones with no empty facts which enable COMP rules. We add an empty fact to the memory of the intruder, swap d with COMP rule, and now there are no such problematic configurations before the rule d in the trace, only one such problematic configuration just after d. We proceed in the obtained trace in the same way, by permuting the leftmost decomposition rule, d', that needs to be moved. By repeating the process we obtain a normalized derivation.

To cover the worst case, we count an additional empty fact each time we swap a decomposition rule with a COMP rule, resulting in at most $(l-1)(k-1)$ additional empty facts.

To obtain the extended normalized derivation we procede the rearrangement of rules by moving SND rules to the end of derivation. Similarly, we need an additional $R(*)$ fact for each application of SND rule. That is bounded by the number of facts in the configuration modulo the intruder because SND rules create N_R facts that intruder cannot consume. There should be at least one $P(*)$ fact in the configuration to enable a SND rule. Therefore the number on SND rules applied is bounded by l. When we add requirements for SND and COMP we can see that additional kl empty facts allow the transformation.

The required number of $P(*)$ facts is there since the original derivation was possible and $P(*)$ facts do not appear in any other rule. □

Remark Notice that a separate memory maintenance theory that includes deletion rules is necessary for normalized derivations of a two-phase theory. By their form, these new deletion rules in the maintenance theories are also decomposition rules, see Figure 4.8. However, if we were to consider the decomposition theory $\mathcal{D}$ with the rules that delete facts, then $\mathcal{D}$ would no longer precede $\mathcal{C}$. For example composition rule GEN enables decomposition rule DELN.

4.3.2 Alternative Definition of Semi-founded Protocol Theory

We include this subsection purely for completeness of the comparison of our theories to those in [18].

In well-founded protocol theories in [18] 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 [18, 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 bounded sub-theory $\mathcal{I}$ similar to [18]. 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 now give this alternative definition and the proof of this claim.

Definition 4.13. A theory $\mathcal{S} \subset \mathcal{T}$ with only balanced rules 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 $\mathcal{T}$.

Initialization theory is a bounded sub-theory, therefore all the facts created by initialization theory are either persistent facts or contain fresh values such as keys associated to the participants.

Definition 4.14. 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 balanced 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 $\mathcal{A}_j$.

Figure 4.10 shows the initialization theory rules for the Needham-Schroeder Protocol. Rules GOODGUY and ANNK announce honest participants' keys while BADGUY and ANNKB do the same for the compromised key pairs. By using the same predicate

$$\begin{aligned}
\text{GOODGUY}: &\quad P(*)P(*) \to \exists k_e.k_d.GoodGuy(k_e,k_d)KP(k_e,k_d)\\
\text{BADKEY}: &\quad P(*)P(*) \to \exists k_e.k_d.BadKey(k_e,k_d)KP(k_e,k_d)\\
\text{ANNK}: &\quad GoodGuy(k_e,k_d)P(*) \to AnnK(k_e)GoodGuy(k_e,k_d)\\
\text{ANNKB}: &\quad BadKey(k_e,k_d)P(*) \to AnnK(k_e)BadKey(k_e,k_d)
\end{aligned}$$

Figure 4.10: Initialization theory for the Needham-Schroeder Protocol.

AnnK for public key announcement, we achieve that honest participants cannot tell the honest public keys from compromised ones.

The next proposition shows that semi-founded 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.

Proposition 4.15. 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

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

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 balanced 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. $\square$

4.4 Standard Protocol Traces

The analysis of security properties of protocols looks for the design flaws and either tries to find them or tries to prove that they cannot occur. Undesirable scenarios include leakage of secret data, such as one's PIN number, or failure in authentication, such as when participants think they are communicating with another participant, but in fact they are communicating with the intruder.

In our model we look for such flaws by trying to find some sort of normalized run which would represent an anomaly. We look at the interaction of the two-phase intruder with bounded memory and the protocol theory in a semi-founded form by using the notion of standard traces, as in [18]. In these traces we allow additional application of memory maintenance rules (rules for deletion of facts), both before and after the intruder's normalized derivations.

Definition 4.16. Given a semi-founded protocol theory $(\mathcal{P}, \mathcal{I})$, where $\mathcal{P} = \mathcal{R} \uplus \mathcal{A}$, a two- phase intruder theory $\mathcal{M}$, and a memory maintenance theory $\mathcal{E}$, a *standard trace* is a derivation that interleaves the steps from $\mathcal{R}$, $\mathcal{A}$ and $\mathcal{E}$ and with normalized derivations from the intruder theory $\mathcal{M}$.

We represent an attack by a goal state in which the intruder learns the secret s, *i.e.*, there is the fact $M_?(s)$ in the configuration. Learning the secret data gives the intruder the power to affect authentication and secrecy of the protocol. If such a secret is a private key of a participant, learning that key would enable intruder to decrypt messages or parts of messages that would otherwise look opaque to him. He could learn private data such as nonces (in the Needham-Schroeder protocol) or tickets for communication with servers (in the Kerberos protocol). Using such information the intruder is able to trick an honest participant, for example, by impersonating the participant whose private key he possesses.

In Section 5.1 we demonstrate that the well-known Lowe anomaly [30] can be carried out in a standard trace by a memory bounded intruder. In Sections 5.2 - 5.7 we also give similar encodings for a number of known anomalies for different protocols.

Chapter 5

Encoding Known anomalies with a Memory Bounded Intruder

We can show that many protocol anomalies, such as Lowe's anomaly [30], can also occur when using our memory bounded adversary. We assume that the reader is familiar with such anomalies, see [14, 18, 30, 8, 10]. In this Section, we demonstrate in detail encoding of Lowe's anomaly of the Needham-Schroeder protocol and encoding of anomalies for other protocols, such as Yahalom [14], Otway-Reese [14, 39], Woo-Lam [14], Kerberos 5 [8, 10] and PKINIT, the public key extension of Kerberos 5.

Table 5.1 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 memory bounded intruder is present. For instance, to realize the Lowe anomaly to the Needham-Schroeder protocol, the intruder requires only seven $R(*)$ facts.[1]

One can interpret the total number of facts needed 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. We believe, therefore, that such values can be used as a quantitative measure on how secure a protocol is.

[1]Notice that here we only encode standard anomalies described in the literature [8, 14, 39]. 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.

Table 5.1: The size of configurations (m), the number of $R(*)$ facts, the size of configurations modulo intruder (l), and the upper-bound on the size of facts (k) needed to encode protocol runs and known anomalies when using LSTSes with balanced actions. The largest size of facts needed to encode an anomaly is the same as in the corresponding normal run of the protocol. In the cases for the Otway-Rees and the Kerberos 5 protocols, we encode different anomalies, which are identified by the numbering, as follows: $^{(1)}$ The type flaw anomaly in [14]; $^{(2)}$ The attack 5 in [39]; $^{(3)}$ The ticket anomaly and $^{(4)}$ the replay anomaly in [8]; $^{(5)}$ The PKINIT anomaly also for Kerberos 5 described in [10].

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

5.1 Lowe Attack to Needham-Schroeder Protocol

We formalize the well known Lowe attack to the Needham-Schoreder protocol [30], informally depicted in Figure 5.1. We show how the intruder uses his two-phase theory and the memory maintenance theory to perform the attack.

$$\begin{array}{l}A \longrightarrow B : \ \{A, n_a\}_{kb} \\ B \longrightarrow A : \ \{n_a, n_b\}_{ka} \\ A \longrightarrow B : \ \{n_b\}_{kb}\end{array}$$

Figure 5.1: Needham-Schroeder Public Key Protocol.

Description of the Needham-Schroeder protocol and its semi-founded theory was given in Section 4.2.1. Differently to the theory shown in Figure 4.6 where the protocol runs with no intruder present and the network is represented by the predicate N, in the theory represented in Figure 5.2 network is modeled by two predicates, N_S and N_R. Participants of the protocol send messages through $N_S(m)$ facts and can read messages from $N_R(m)$ facts. It is the intruder who controls the network by transforming $N_S(m)$ facts into $N_R(m)$ facts. In other words, intruder is the network and the protocol cannot run without the intruder (network).

Role Regeneration Theory :

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

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

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

Figure 5.2: Semi-founded protocol theory for the Needham-Schroeder Protocol.

$$\begin{tikzcd}
A \arrow[r, "\{A, n_a\}_{K_B}"] & B \arrow[r, "\{A, n_a\}_{K_C}"] & C \\
A & B \arrow[l, "\{n_a, n_c\}_{K_A}"'] & C \arrow[l, "\{n_a, n_c\}_{K_A}"'] \\
A \arrow[r, "\{n_c\}_{K_B}"] & B \arrow[r, "\{n_c\}_{K_C}"] & C
\end{tikzcd}$$

Figure 5.3: Lowe attack to Needham-Schroeder Protocol

The informal description of Lowe's anomaly is depicted in Figure 5.3. This attack 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 who is very happy to talk to Alice. 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.

This attack shows two main points of insecurity of this protocol. Firstly, 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 authentification. 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.

$$\begin{aligned}
W_I = \quad & GoodGuy(k_{e1}, k_{d1})\ KP(k_{e1}, k_{d1})\ AnnK(k_{e1}) \\
& BadKey(k_{e2}, k_{d2})\ KP(k_{e2}, k_{d2})\ AnnK(k_{e2}) \\
& GoodGuy(k_{e3}, k_{d3})\ KP(k_{e3}, k_{d3})\ AnnK(k_{e3}) \\
& M_{ke}(k_{e1})\ M_{ek}(k_{e2})\ M_{dk}(k_{d2})\ M_{k3}(k_{e3})
\end{aligned}$$

A trace representing the anomaly is shown below.

Alice starts the protocol by sending the message to Bob, but the intruder intercepts it.

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

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.

$\rightarrow_{DEC}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ D(\langle n_a, k_{e1}\rangle)\ M_c(enc(k_{e2}, \langle n_a, k_{e1}\rangle))\ R(*)P(*) \rightarrow_{DM}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M_s(\langle n_a, k_{e1}\rangle)\ M_c(enc(k_{e2}, \langle n_a, k_{e1}\rangle))\ R(*)P(*) \rightarrow_{DELMC}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M_s(\langle n_a, k_{e1}\rangle)\ B(*)\ R(*)P(*) \rightarrow_{USES}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M_s(\langle n_a, k_{e1}\rangle)\ C(\langle n_a, k_{e1}\rangle)\ B(*)P(*) \rightarrow_{ENC}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M_s(\langle n_a, k_{e1}\rangle)\ C(enc(k_{e3}, \langle n_a, k_{e1}\rangle))\ B(*)P(*) \rightarrow_{SND}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ M_s(\langle n_a, k_{e1}\rangle)\ N_R(enc(k_{e3}, \langle n_a, k_{e1}\rangle))\ B(*)R(*) \rightarrow_{DELS}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ N_R(enc(k_{e3}, \langle n_a, k_{e1}\rangle)\ B(*)R(*)R(*) \rightarrow_{DELB}$
$W_I A_1(k_{e1}, k_{e2}, n_a)\ B_0(k_{e3})\ N_R(enc(k_{e3}, \langle n_a, k_{e1}\rangle)\ R(*)R(*)R(*) \rightarrow$

Additionally intruder deletes some facts from his memory. 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.

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

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

$\rightarrow_{A2}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ N_S(enc(k_{e2}, n_c))\ R(*)R(*)R(*) \rightarrow_{REC}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ D(enc(k_{e2}, n_c))\ R(*)R(*)P(*) \rightarrow_{DEC}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ M_c(enc(k_{e2}, n_c))\ D(n_c)\ R(*)P(*) \rightarrow_{DELMC}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ B(*)\ D(n_c)\ R(*)P(*) \rightarrow_{LRNN}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ B(*)\ M_n(n_c)\ R(*)P(*) \rightarrow_{USEN}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ B(*)\ M_n(n_c)\ C(n_c)\ P(*) \rightarrow_{ENC}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ B(*)\ M_n(n_c)\ C(enc(k_{e3}, n_c))\ P(*) \rightarrow_{SND}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ B(*)\ M_n(n_c)$
$\quad N_R(enc(k_{e3}, n_c))\ R(*) \rightarrow_{(DELN,DELB)}$
$W_I A_2(k_{e1}, k_{e2}, n_a, n_c)\ B_1(k_{e3}, k_{e1}, n_a, n_c)\ N_R(enc(k_{e3}, n_c))\ R(*)R(*)R(*) \rightarrow$

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

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

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

The attack requires a configuration of at least 19 facts in total: 12 $P(*)$ facts (for the honest participants) and 7 $R(*)$ facts (for the intruder). The size of facts has to be at least 6, to be able to represent the largest message sent in the protocol.

5.2 Yahalom Protocol

The informal description of the Yahalom protocol is given in figure 5.4. The protocol has been shown to be flawed by several authors.

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

Figure 5.4: 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. 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 5.5.

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 the role states and another fact for the network predicate. The largest message sent is the message that the server S sends to A and it has 15 symbols. 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.

Role Regeneration Theory :

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

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

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

Figure 5.5: Semi-founded protocol theory for the Yahalom Protocol.

5.2.1 An Attack on Yahalom Protocol

An anomaly on the Yahalom protocol is shown in Figure 5.6.
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.

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

Figure 5.6: An attack on Yahalom Protocol.

Initial set of facts for the anomaly is:

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

For the symmetric encryption and decryption intruder uses ENCS and DECS rules. This attack requires encryption with a composed key so intruder needs ENCM rule for such encyption: ENCM : $C(x)C(y) \to M_k(x)C(enc(x, y))$.

The trace with the anomaly is shown below.

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

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.

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

Intruder intercepts the message intended for the server.

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

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.

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

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

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

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

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

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

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

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.

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

Since the attack assumes that the intruder knows the key k_{BS} shared between the server and Bob, so another memory fact, $M_k(k_{BS})$, stores that key. As shown above, 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. As per messages exchanged, the size of the facts should be at least 14.

5.3 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 5.7.

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

Figure 5.7: 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). 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.

A semi-founded protocol theory for Otway-Rees protocol is given in Figure 5.8.

In a normal protocol run, 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 : $Guy(G, k_{GS})\ AnnN(G)\ P(*) \rightarrow Guy(G, k_{GS})\ AnnN(G)\ A_0(k_{GS})$
ROLB : $Guy(G, k_{GS})\ AnnN(G)\ P(*) \rightarrow Guy(G, k_{GS})\ AnnN(G)\ B_0(k_{GS})$
ROLS : $P(*) \rightarrow\ S_0()$
ERASEA : $A_2(k, G, x, y, k') \rightarrow P(*)$
ERASEB : $B_2(k, G, x, y, z, w, k') \rightarrow P(*)$
ERASES : $S_1(G, G') \rightarrow P(*)$

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

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

Figure 5.8: Semi-founded protocol theory for the Otway-Rees Protocol.

5.3.1 A Type Flaw Attack on Otway-Reese Protocol

In this anomaly, shown in Figure 5.9, 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.

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

Figure 5.9: 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" $\langle M, A, B\rangle$ 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.

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

$$
\begin{aligned}
&\to_{USEN}\\
&WA_1(k_{AS}, B, M, n_a)\ M_n(M)\ C(M)\ B(*)R(*)P(*)\\
&\quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \to_{USEC}\\
&WA_1(k_{AS}, B, M, n_a)\ M_n(M)\ C(M)\ \ B(*)P(*)\\
&\quad M_s(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle))\ C(enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)) \to
\end{aligned}
$$

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

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

As has been just shown, this attack requires a configuration of at least 11 facts in total: 6 $P(*)$ facts (for the honest participants) and 5 $R(*)$ facts (for the intruder). Intruder needs 2 $R(*)$ facts to learn participants' names.
The size of facts has to be at least 15. 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 signature. Since we allow only atomic keys within our typed signature this attack is not possible. The tuple of terms $\langle M, A, B\rangle$ cannot be confused with a term of type "key".

5.3.2 Replay Attack on Otway-Rees Protocol

This attack, shown in figure 5.10, 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.

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

Figure 5.10: Replay attack on Otway-Rees Protocol.

As shown in figure 5.10, 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}.

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.

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

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

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

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 $z = (enc(k_{AS}, \langle n_a, \langle M, \langle A, B\rangle\rangle\rangle)$.

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

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

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

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.

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

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{array}{l}
\to_{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(*) \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(*)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(*) \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.

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

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.

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

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.

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

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}.

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

5.4 Woo-Lam Protocol

The informal description of this one-way authentication protocol is shown in Figure 5.11 and a semi-founded protocol theory is given in Figure 5.12.

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

Figure 5.11: 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.

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 5.12: Semi-founded protocol theory for the simplified Woo-Lam Protocol.

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.

5.4.1 An Attack on Simplified Woo-Lam Protocol

The Woo-Lam protocol in its various versions appear to be subject to various attacks. We present one such anomaly in Figure 5.13.

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

Figure 5.13: 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: $W = Guy(A, K_{AS})\ Guy(B, k_{BS})\ AnnN(A)\ AnnN(B)$.

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

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

5.5 An Audited Key-distribution Protocol

The following protocol was introduced in [18]. 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.

Figure 5.14 shows the protocol for $s = 4$ and its semi-founded protocol theory is given in Figure 5.15. 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. It has the side-effect of broadcasting the SECRET if C receives the error message.

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 5.14: Exponential Protocol

If no intruder is present the protocol consists in $2s + 1$ messages, ends by OK message and the SECRET if not revealed. Initial set of facts represents key distribution for communication with the server and includes additional 4 facts representing principals' names. One fact is required for the network predicate. 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 Theories :

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

Protocol Theories :

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

Figure 5.15: Protocol theory rules in semi-founded form

Role regeneration theory optimizes the number of facts required by deleting final role states with ERASE rules. It is therefore possible to run the protocol with only 2 role state predicates in a configuration at a time, *i.e.* with no more than one protocol session running in parallel.

Therefore, a protocol run between A, B_1, B_2, B_3, B_4 and C with no intruder involved requires a configuration of at least 11 facts of the size of at least 10.

5.5.1 An Exponential Attack

It is argued in the [18] 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.

5.6 Symmetric Key Kerberos 5 Protocol

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 in [8]. 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 Figure 5.16. A semi-founded protocol theory is given in Figure 5.17.

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

Figure 5.16: Kerberos 5 Protocol.

A standard 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.

Role Regeneration Theory :

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

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

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

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

Figure 5.17: Semi-founded protocol theory for the Kerberos 5 Protocol.

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.

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.

The standard trace representing the normal protocol run is given below:

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

5.6.1 Ticket Anomaly in Kerberos 5 Protocol

The informal description of the ticket anomaly in Kerberos 5 protocol is given in Figure 5.18. 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 5.18: 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.

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

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.

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

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

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.

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

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.

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

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

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

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.

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

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 $Y = enc(k_S,\langle SKey,C\rangle)$.

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

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

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.

5.6.2 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.

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

Figure 5.19: Replay anomaly of Kerberos 5 Protocol

We will model the replay of the third request message from the protocol as shown in Figure 5.19.

Intruder basically observes the protocol run remembering the request message to the Server. He starts by only forwarding the network predicates, *i.e.* transforms the N_S to N_R predicate. The last request message is digested with the data from the request message kept in intruder's memory for later replay. If the original requests was accepted by$S$, so may be the replayed one as well. In that case the server S generates fresh credentials based on replayed requests. 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:

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

The standard trace of the anomaly is given below:

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

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

$$
\begin{aligned}
& \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) \\
& \quad 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(*) \\
& \quad N_R(\langle C, \langle T, n_1\rangle\rangle)\ Valid_K(C, T, n_1) \\
& \quad 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(*) \\
& \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) \\
& \quad R(*)R(*)R(*)R(*) \rightarrow_{FWD} \\
& W\ C_1(C, T, n_1)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*)P(*)P(*) \\
& \quad 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) \\
& \quad 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(*) \\
& \quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
& \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) \\
& \quad 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(*) \\
& \quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey)\ Valid_T(C, S, n_2) \\
& \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) \\
& \quad 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(*) \\
& \quad Auth(enc(k_T, \langle AKey, C\rangle), T, AKey) \\
& \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) \\
& \quad R(*)R(*)R(*)R(*) \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
\end{aligned}
$$

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

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

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

Intruder starts composing the message.

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

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

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

Intruder keeps the data necessary for the request replay in his memory.

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

Again intruder only forwards the message.

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

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.

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

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.

5.7 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. 5.20, 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 5.21.

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

Figure 5.20: PKINIT Protocol.

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:

$$
\begin{array}{rl}
W = & Client(C, pk_C)\ KP(pk_C, sk_C)\ AnnK(pk_C) \\
& KAS(K)\ KP(pk_K, sk_K)\ AnnK(pk_K) \\
& TGS(T)\ TGSKey(T, k_T) \\
& Server(S)\ ServerKey(S, k_S)\ .
\end{array}
\tag{5.1}
$$

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). 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.

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 : $C_1(C, T, n_1, n_2, t_C)\ Server(S)\ P(*)$
$\quad N_S(\langle enc(pk_C, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle),$
$\quad\quad \langle C, \langle X, enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle\rangle)$
$\quad \rightarrow \exists n_3.C_2(C, T, S, AKey, n_3)\ Server(S)\ Auth(X, T, AKey)$
$\quad N_S(\langle X, \langle enc(AKey, C), \langle C, \langle S, n_3\rangle\rangle\rangle\rangle)$
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 : $K_0(K)\ Client(C, pk_C)\ TGSKey(T, k_T)\ Valid_K(C, T, n_1)\ Clock_K(t_K)$
$\quad N_R(\langle Cert_C, \langle enc(sk_C, \langle t_C, n_2\rangle), \langle C, \langle T, n_1\rangle\rangle\rangle\rangle)$
$\quad \rightarrow \exists k.AKey.K_1(K)\ Client(C, pk_C)\ TGSKey(T, k_T)\ P(*)P(*)$
$\quad N_S(\langle enc(pk_C, \langle Cert_K, enc(sk_K, \langle k, n_2\rangle)\rangle),$
$\quad\quad \langle C, \langle enc(k_T, \langle AKey, C\rangle), enc(k, \langle AKey, \langle n_1, \langle t_K, T\rangle\rangle\rangle)\rangle\rangle\rangle)$
T1 : $T_0(T)\ TGSKey(T, k_T)\ ServerKey(S, k_S)\ Valid_T(C, S, n_2)$
$\quad N_R(\langle enc(k_T, \langle AKey, C\rangle), \langle enc(AKey, C), \langle C, \langle S, n_2\rangle\rangle\rangle\rangle)$
$\quad \rightarrow \exists SKey.T_1(T)\ TGSKey(T, k_T)\ SerevrKey(S, k_S)\ P(*)$
$\quad N_S(\langle C, \langle enc(k_S, \langle SKey, C\rangle), enc(AKey, \langle SKey, \langle n_2, S\rangle\rangle)\rangle\rangle)$
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 5.21: Semi-founded protocol theory for the PKINIT

The standard 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(*) \to_{clock_C}\\
W\ C_0(C)\ K_0(K)\ T_0(T)\ S_0(S)\ Clock_C(t_C)\ P(*)P(*)P(*) \to_{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(*) \to_{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(*) \to_{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(*) \to_{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\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) \to_{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) \to_{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) \to_{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) \to_{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) \to_{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)) \to_{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)) \to_{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)) \to_{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}$$

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 validitiy of relevant Kerberos messages. They are determined by an external process and we represent them with separate rules:

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

5.7.1 Man-in-the-middle Attack on PKINIT

A Man-in-the-middle attack on PKINIT is informally shown in Figure 5.22. 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.

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

Figure 5.22: 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:

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

The standard trace representing this anomaly is given below:

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

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

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

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.

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

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)\\
&\quad R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*)P(*)\\
&\quad D(\langle enc(pk_B,\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle),\\
&\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(*)\\
&\quad D(enc(pk_B,\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle))\ R(*)R(*)R(*)R(*)\\
&\quad 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_{DEC}\\
&W\ C_1(C,T,n_1,n_2,t_C)\ K_1(K)\ T_0(T)\ S_0(S)\ M_g(C)\\
&\quad R(*)R(*)R(*)P(*)P(*)P(*)P(*)\\
&\quad 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)\\
&\quad 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)\\
&\quad R(*)R(*)R(*)P(*)P(*)P(*)P(*)B(*)\ D(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)\\
&\quad 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(*)\\
&\quad B(*)\ M_s(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)\ R(*)R(*)R(*)\\
&\quad 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(*)\\
&\quad M_s(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)\ R(*)R(*)R(*)\\
&\quad 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(*)\\
&\quad M_s(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)\ R(*)R(*)R(*)\\
&\quad 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(*)\\
&\quad M_s(\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle)\ R(*)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) \rightarrow
\end{aligned}$$

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

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

$$
\begin{array}{l}
\to_{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(*) \\
\quad M_s(\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(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) \to_{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 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) \to_{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) \to_{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) \to_{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), \\
\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) \to_{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) \\
\quad R(*)R(*)R(*)R(*)R(*)R(*)P(*)P(*)P(*) \\
\quad N_R(\langle enc(pk_C,\langle Cert_K, enc(sk_K,\langle k,n_2\rangle)\rangle), \\
\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) \to
\end{array}
$$

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

$$
\begin{array}{l}
\to_{C2} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ P(*)P(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad N_S(\langle C,\langle\langle enc(k_T,\langle AKey,C\rangle), enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle\rangle) \to_{constraint_T} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Valid_T(C,S,n_2)\ P(*) \\
\quad N_S(\langle C,\langle\langle enc(k_T,\langle AKey,C\rangle), enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle\rangle) \to_{FWD} \\
W\ C_2(C,T,S,AKey,n_2)\ K_1(K)\ T_0(T)\ S_0(S)\ R(*)R(*)R(*)R(*)R(*)R(*) \\
\quad Auth(enc(k_T,\langle AKey,C\rangle),T,AKey)\ Valid_T(C,S,n_2)\ P(*) \\
\quad N_R(\langle C,\langle\langle enc(k_T,\langle AKey,C\rangle), enc(k_C,\langle AKey,\langle n_1,T\rangle\rangle)\rangle\rangle\rangle) \to
\end{array}
$$

$$
\begin{array}{l}
\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_{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}
$$

There should be 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.

Chapter 6

Conclusions, Related and Future Work

6.1 Related Work

As previously discussed, we build on the framework described in [26, 27]. In particular, we introduce the notion of progressing collaborative systems and investigate the use of actions that can create nonces. We tighten the lower bounds from [26, 27] by using the progressing assumption, and we also provide new results for when nonces can be created. In [5, 6, 28], a temporal logic formalism for modeling collaborative system is introduced. In this framework, one relates the scope of privacy to the specific roles of agents in the system. For instance, in our medical scenario, the patient's test results, which normally should not be accessible to any agent, are accessible to the agent that has the role of the patient's doctor. We believe that our system can be adapted or extended to accommodate such roles depending on the scenario considered. In particular, the health insurance scenario discussed in [28] has many connections with our medical scenario and it seems possible to implement it in our framework.

In [22] Harrison *et al.* present a formal approach to access control. In their proofs, they faithfully encode a Turing machine in their system. However, differently from 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 nonces, then 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 it is NP-complete, but in their proof they implicitly impose a bound on the number of nonces that can be created. Although related to our case with LSTSes containing monadic actions, their result is different from ours since they do not add the notions of progressing systems nor of balanced actions to their system.

Our paper is closely related to frameworks based on multiset rewriting systems used to specify and verify security properties of protocols [1, 2, 12, 15, 18, 36]. While here we are concerned with systems where agents are in a *closed room* and collaborate, there, 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, copy messages. On the other hand, we assume here that each agent has a bounded memory, technically imposed by the use of balanced actions. Therefore, the lower bounds obtained here are tighter than the results obtained in those papers.

Much work on reachability related problems has been done within the Petri nets (PNs) community, see *e.g.*, [19]. Specifically, we are interested in the *coverability problem* which is closely related to the partial goal reachability problem in LSTSes [26]. To our knowledge, no work that captures exactly the conditions in this paper has yet been proposed. For instance, [19, 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.

6.2 Conclusions and Future Work

In this paper we introduced an important class of collaborating systems called progressing collaborative systems. These systems seem to capture well many administrative processes, namely, those in which the same action does not need to be performed more than once. We obtain exact lower bounds for the weak plan compliance problem when using such systems under different conditions, tightening results from the literature. However, we have already expressed our doubts on our current approach to progressing. Although we were able to get an improvement in the complexity of some problems, we found progressing with nonces quite complicated and unclear. We are currently working on an alternative notion of progressing which would provide more efficient systems.

We extend existing models and also investigate systems with actions that can create nonces. Introducing nonces increases the expressivity of our systems. Many real life processes involve fresh values, such as processes that involve unique identification, *e.g.* bank transactions or security protocols. We use focused proofs in linear logic to formalize the operational semantics of such actions. We provide various complexity results for the three types of compliance problems for these systems. In particular, using balanced actions and bounding the size of facts, we concentrate on systems with a fixed memory.

As an application of our results, we turn to security protocol analysis and the complexity of the secrecy problem. We also 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.

There are many interesting directions to follow from this work, some of which we are already pursuing. For instance, we are further extending the expressivity of our systems by adding the explicit time. One the other hand, we are comparing the power of memory bounded intruder with the standard Dolev-Yao intruder with unbounded memory, by providing an upper bound on the memory needed by a memory bounded intruder to find an attack on a given protocol. Also, we believe that, despite of our idealized model, the numbers appearing in Table 5.1 provide some measure on the security of protocols. In general, we seek to provide further quantitative information on the security of protocols. Some of these parameters appear in existing model checkers, such as Murϕ. We are investigating precise connections to such tools. In particular, we would like to understand better the impact of our work to existing protocol analysis tools, in particular, our PSPACE upper-bound result. On the implementation side, we hope to provide the means to model-check systems with actions that can create nonces. It is also interesting to leverage the work in [5, 6, 28] and specify policies in temporal logic, instead of intuitionistic logic. We expect to do so in the near future.

Bibliography

[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-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[4] David Baelde, Andrew Gacek, Dale Miller, Gopalan Nadathur, and Alwen Tiu. The Bedwyr system for model checking over syntactic expressions. In Frank Pfenning, editor, *cade07*, number 4603, pages 391–397. Springer, 2007.

[5] 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.

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

[7] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov. A formal analysis of some properties of Kerberos 5 using MSR. University of Pennsylvania, Department of Computer and Information Science, Technical Report MS-CIS-04-04, April 2004, 59 pages

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

[9] Iliano Cervesato, Nancy A. Durgin, Patrick Lincoln, John C. Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *CSFW*, pages 55–69, 1999.

[10] I. Cervesato, A. D. Jaggard, A. Scedrov, J.-K. Tsay, and C. Walstad. Breaking and fixing public-key Kerberos. *Inf. Comput.*, 206(2-4):402–424, 2008.

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

[12] Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 261, Washington, DC, USA, 2003. IEEE Computer Society.

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

[14] John Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 1997.

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

[16] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, New York, NY, USA, 1971. ACM.

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

[18] 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.

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

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

[21] J. Goguen. Order sorted algebra. Technical Report 14. Semantics and Theory of Computation Series, UCLA Computer Science Department, 1978.

[22] 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.

[23] M. Kanovich. The direct simulation of Minsky machines in linear logic. *In J.-Y. Girard, Y. Lafont,and L. Regnier, editors, Advances in Linear Logic, volume 222 of London Mathematical Society Lecture Notes*, pages 123145, 1995.

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

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

[26] 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.

[27] 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.

[28] 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.

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

[31] Jonathan K. Millen A Necessarily Parallel Attack. In *In Workshop on Formal Methods and Security Protocols*, 1999.

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

[33] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993999, 1978.

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

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

[36] 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.

[37] Konstantinos Sagonas, Terrance Swift, David S. Warren, Juliana Freire, Prasad Rao, Baoqiu Cui, Ernie Johnson, Luis de Castro, Rui F. Marques, Steve Dawson, and Michael Kifer. *The XSB Version 3.0 Volume 1: Programmer's Manual*, 2006.

[38] Alwen Tiu. Model checking for π-calculus using proof search. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653, pages 36–50. Springer, 2005.

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

Summary

In this thesis we consider existing models for collaborative systems with confidentiality policies. We extend these systems with nonce creation. We also consider a restriction that each instance of an action is used at most once in a process. Administrative processes usually involve such progressing behavior, that is, whenever a transaction is performed, it does not need to be repeated. We investigate the complexity of the decision problem whether there exists a sequence of transitions from an initial state to a final state that avoids any critical states, e.g., states which conflict with the given confidentiality policies. Collaborative systems allow modeling of the protocols and the relevant security problems. We model systems with limited resources to allow a new view on the standard Dolev-Yao intruder, namely, an intruder with bounded memory. The power of such an intruder is analyzed.

Sažetak

Ovaj se rad zasniva na modelima za sustave suradnje (collaborative systems) sa strategijama povjerenja (confidentiality policies). Ti se modeli proširuju uvodjenjem novih ili svježih vrijednosti (nonce). Proučavaju se svojstva sustava uz ograničenje da se svaka instanca pravila može koristiti najviše jedanput u procesu. Administrativni procesi obično imaju takvo svojstvo, tj. provedena akcija ne treba se ponoviti. Istažiti ćemo složenost problema postojanja plana, tj. niza akcija od nekog početnog do završnog stanja koji ne vodi do kritičnih stanja, odnosno stanja koja su u suprotnosti sa strategijom povjerenja. Sustavi za suradnju pogodni su za modeliranje protokola i njihovih sigurnosnih svojstava. Balansirani sustavi modeliraju ograničene resurse, pa omogućavaju novi pogled na napadača na protokole. Za razliku od standardnog Dolev-Yao intrudera modelira se napadač s ograničenom memorijom i proučava njegova snaga.

Curriculum Vitae

Tajana Ban Kirigin was born in 1972. in Rijeka, Croatia. She graduated at the Faculty of Arts and Sciences, University of Rijeka in 1996, as a professor of mathematics and computer science, with the thesis "Coherence Spaces and Gödel's System T" supervised by dr.sc. Dean Rosenzweig. In the same year she became a graduated student of mathematics, at the Department of Mathematics, Faculty of Natural Sciences and Mathematics, University of Zagreb. She defended Master Thesis "Higher order logic and system Isabelle", supervised by dr.sc Dean Rosenzweig, in 2004, at the same Department of Mathematics, Faculty of Natural Sciences and Mathematics, University of Zagreb.

Her main research interests are privacy in collaborative systems, computability and recursive theory, computational dificulty of problems.

From 1999. she has been working as a teaching assistant at the Faculty of Arts and Sciences and at the Department of Mathematics, University of Rijeka. In particular, she has been the professor for the courses in Mathematical Logic and History of Mathematics.

She is a member of Seminar of mathematical logic and Seminar for Computer science at Department of Mathematics, Faculty of Natural Sciences and Mathematics, University of Zagreb.