Strategy for Flaws Detection based on a Services-driven Model for Group Protocols

Najah Chridi, Laurent Vigneron

Strategy for Flaws Detection based on a Services-driven Model for Group Protocols

Najah Chridi and Laurent Vigneron *

LORIA – UHP-UN2 (UMR 7503)
BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France
{chridi,vigneron}@loria.fr

Abstract. Group key agreement is important in many modern public and dedicated applications. Nevertheless, as they have to be secure, their design is not straightforward. As such, the modelling and the verification of such protocols are necessary in order to avoid eventual weaknesses. This paper investigates a strategy for flaws detection for group protocols properties. The strategy is based on both a services driven model for group protocols and constraint solving. Our strategy has been applied to several group protocols such as GDH.2 and the Asokan-Ginzboorg protocol. This permits to pinpoint new attacks on them. The result found for the case of GDH.2 with four participants can be generalized to n participants. Another general attack has also been found for the case of the A-GDH.2 protocol.

1 Introduction

In recent years, applications requiring an unbounded number of participants have received increasing attention either for public domains or dedicated ones. As such, the design of secure group protocols [14] continues to be one of the most challenging areas of security research. To secure their communications, group members need to use a shared key, known as group key, which has to be updated following the dynamics of the group (join or leave operations, ...). Therefore, several protocols dedicated to key establishment and updates have been proposed [8]. Among them, we are particularly interested in group key agreement protocols [9]. These protocols enable a group of participants to share a common key over insecure public networks, even when adversaries completely control all the communications.

Research in formal verification of cryptographic protocols has so far mainly concentrated on reachability properties such as secrecy and authentication. It has given so successful interesting results in the last years that this field could be considered as saturated. As such, many fully automatic tools have been developed and successfully applied to find flaws in published protocols, where many of these tools employ so-called *constraint solving* (see, e.g., [3]). Nowadays, dealing with the verification of group protocols arises several problems. Indeed, such protocols highlight new requirements and consider some complicated intended security properties other than secrecy and authentication. In fact, most of the verification approaches can only tackle specific models of protocols, and most of the time require the size of the group to be set in advance. This leads to the restriction of the chances to discover attacks. Besides, group membership is very dynamic; participants can join or leave the group at any time. As such, security requirements are more complicated to satisfy.

The main contribution of the present work is a strategy for flaws detection for the group protocols security properties. Our approach is based on both the services driven model described

* This work is supported by the QSL operation COWS.

in [4] and constraint solving. As mentioned, constraint solving has been successfully employed for reachability properties in the past and proved to be a good basis for practical implementations. The services driven model permits to specify security properties for group protocols as sets of constraints. This model specifies both group protocols and a large class of their intended properties, varying from standard secrecy and authentication properties to much trickier ones, such as key integrity, and backward and forward secrecy. Hence, our strategy paves the way for extending existing tools for reachability properties to deal with security properties for group protocols.

In this paper, we focus on group key establishment protocols. But this is worth mentioning that our method is also dealing with contributing protocols. To present our results, the paper is structured as follows. We first introduce our running example: The Asokan-Ginzboorg Protocol [2] (Section 2). This protocol will be used throughout this paper to illustrate every new notion introduced. In Section 3, we present the input required by our method. Then, Section 4 provides the necessary background concerning the services driven model. In Section 5, we show how this model can be used to search for attacks. The management of constraints and intruder knowledge is explained in Sections 6 and 7. We illustrate the application of our method to two examples in Section 8. And after a comparision with related work (Section 9), we summarize the results obtained by our approach and then discuss the related work (Section 10).

2 Running Example: The Asokan-Ginzboorg Protocol

Throughout this paper we will illustrate our ideas using a running example: the Asokan-Ginzboorg protocol [2]. It describes the establishment of a session key between a leader (A_n) and a random number n of participants (A_i where $1 <= i <= n$). The protocol proceeds by assuming that a short group password P is chosen and displayed, and then known by all. We assume also that there are two informations known by all the group: a one way hash function H and a commonly known function F. When the leader starts the execution of the protocol by sending the key of encoding (E), each participant generates two informations (a symmetric key (R_i) and a contribution to the group key (S_i)) and sends them encrypted by the key E. Messages exchanged throughout the protocol are expressed as follows:

$$
\begin{array}{ll}
1.A_n \longrightarrow \text{ALL} & : A_n, \{E\}_P \\
2.A_i \longrightarrow A_n & : A_i, \{R_i, S_i\}_E, \text{ i=1,...,n-1} \\
3.A_n \longrightarrow A_i & : \{\{S_j, \text{j=1,...,n}\}\}_{R_i}, \text{ i=1,...,n-1} \\
4.A_i \longrightarrow A_n & : A_i, \{S_i, H(S_1, \ldots, S_n)\}_k \text{ some } i, k = F(S_1, \ldots, S_n).
\end{array}
$$

In this messages exchange, E is a public key generated by the leader and used to encrypt the contribution (S_i) of each participant A_i. R_i denotes a fresh symmetric key generated by the participant A_i, sent to the leader with the contribution S_i. The leader will use it to encrypt all contributions (including S_n) in order to send the whole message to the participant A_i.

3 The Method's Input

As any communication protocol, a group protocol can be seen as an exchange of messages between several participants. This exchange is usually described by the set of actions executed by each participant in a normal protocol execution, i.e. without intervention of an Intruder.
Formally speaking, we define an instance of the protocol as the union of instances of roles and Intruder knowledge. An instance of a protocol is then given by $(\{\mathcal{R}_p \rightarrow \mathcal{S}_p\}_{p \in \mathcal{P}}, <_{\mathcal{P}}, S_0)$ where, $\mathcal{P}$ is a finite set and:

– $\{\mathcal{R}_p \rightarrow \mathcal{S}_p\}_{p\in\mathcal{P}}$ denotes the set of rules of receive-send messages exchanged between honest participants. Each rule defines one step of the protocol: the messages sent by a honest participant ($\mathcal{R}_p$) and the expected response ($\mathcal{S}_p$). Note that $\mathrm{Var}(\mathcal{R}_p) \subseteq \mathrm{Var}(\mathcal{S}_p)$.
– $<_{\mathcal{P}}$ is a partial order over $\mathcal{P}$.
– S_0 is a set of terms representing the initial Intruder knowledge.

Let us return to our running example: the Asokan-Ginzboorg protocol. Having tested our method over several scenarios of this protocol, we have found an interesting result in the case of two parallel sessions. Thus, the modelling considered in the following corresponds to that scenario. In the first session, we have two participants: A_1 is the leader and A_2 is a normal member of the group. In the second session, the roles are exchanged.
Throughout this paper, while expressing informations related to the Asokan-Ginzboorg protocol, we adopt the following notations:

– p_{ijk} denotes the j-th step of the protocol played by the i-th participant in the k-th session.
– Alg_{ij} the point of vue of the group key of the i-th participant during the j-th session.
– Terms written in capital letters are informations known by the participants whether from the beginning of the execution or generated through the execution.
– Terms written in small letters denote variables. They may be instantiated by any values of the same type.
– S_{ij} denotes the contribution to the group key of the i-th participant in the j-th session.

The two sessions are expressed by the following steps:

$$
\begin{array}{lrll}
\mathbf{p_{111}}: & Init & \longrightarrow A_1, \{E_1\}_P & \\
\mathbf{p_{121}}: & x_1, \{x_2, x_3\}_{E_1} & \longrightarrow \{x_3, S_{11}\}_{x_2} & \\
\mathbf{p_{131}}: & x_1, \{x_3, H(x_3, S_{11})\}_{F(x_3,S_{11})} & \longrightarrow End, & \mathbf{Alg_{11} = F(x_3, S_{11})} \\
\mathbf{p_{211}}: & x_4, \{x_5\}_P & \longrightarrow A_2, \{R_1, S_{21}\}_{x_5} & \\
\mathbf{p_{221}}: & \{x_6, x_7\}_{R_1} & \longrightarrow A_2, \{S_{21}, H(x_6, x_7)\}_{F(x_6,x_7)}, & \mathbf{Alg_{21} = F(x_6, x_7)} \\
 & & & \\
\mathbf{p_{212}}: & Init & \longrightarrow A_2, \{E_2\}_P & \\
\mathbf{p_{222}}: & x_8, \{x_9, x_{10}\}_{E_2} & \longrightarrow \{x_{10}, S_{22}\}_{x_9} & \\
\mathbf{p_{232}}: & x_8, \{x_{10}, H(x_{10}, S_{22})\}_{F(x_{10},S_{22})} & \longrightarrow End, & \mathbf{Alg_{22} = F(x_{10}, S_{22})} \\
\mathbf{p_{112}}: & x_{11}, \{x_{12}\}_P & \longrightarrow A_1, \{R_2, S_{12}\}_{x_{12}} & \\
\mathbf{p_{122}}: & \{x_{13}, x_{14}\}_{R_2} & \longrightarrow A_1, \{S_{12}, H(x_{13}, x_{14})\}_{F(x_{13},x_{14})}, & \mathbf{Alg_{12} = F(x_{13}, x_{14})}
\end{array}
$$

The set of steps $\mathcal{P} = \{p_{111}, p_{121}, p_{131}, p_{211}, p_{221}, p_{212}, p_{222}, p_{232}, p_{112}, p_{122}\}$ is ordered by the partial order $<_{\mathcal{P}}$: $p_{111} <_{\mathcal{P}} p_{121} <_{\mathcal{P}} p_{131}$, $p_{211} <_{\mathcal{P}} p_{221}$, $p_{212} <_{\mathcal{P}} p_{222} <_{\mathcal{P}} p_{232}$and $p_{112} <_{\mathcal{P}} p_{122}$.

4 The Services driven Model for Group Protocols

The services driven model presented in [4] permits to model contributed protocols, to study their characteristics and security properties. This model has been applied on several protocols, such as A-GDH.2, SA-GDH.2, Asokan-Ginzboorg and Bresson-Chevassaut-Essiari-Pointcheval and has permitted to pinpoint several existing attack types on them. A group protocol is modelled by three components $<\mathcal{A}, \mathcal{K}, \mathcal{S}>$, where

– $\mathcal{A}$: set of agents, members of the group,
– $\mathcal{K}$: set of members knowledge,
– $\mathcal{S}$: set of services. A *service* denotes the *contribution* of a participant to the generation of the group key. The contribution of an agent a_i to an agent a_j is all information (message) generated by a_i and necessary for a_j to deduce the group key.

Let $i \in \mathbb{N}$, each $\mathbf{a_i} \in \mathcal{A}$ (*i-th* participant) is linked with other sets defined as follows:

- $\mathbf{S_i} \subseteq \mathcal{S}$ is the *minimal* set of services necessary to a_i in order to generate the group key;
- $\mathcal{K}_\mathbf{i} \subseteq \mathcal{K}$ is the *minimal* set of *private* knowledge of a_i, useful for generating services and the group key; it includes the initial private knowledge and the information generated during the protocol's execution.
- $\mathcal{K}_\mathbf{ij} \subseteq \mathcal{K}$ is the set of knowledge *shared* between agents a_i and a_j; it denotes the *minimal* set of shared knowledge that is useful for generating the group key; this information is given by the protocol's specification. Note that $\mathcal{K}_{ij} = \mathcal{K}_{ji}$.

In addition to the services used for the group key generation, other subsets of services representing the services provided by an agent are defined. Thus, $\mathbf{S_{a_i}}$ denotes the subset of services to which a_i has contributed (directly or not) by providing a private information.
$S_{a_i} = \{s \in \mathcal{S} \mid \exists t \text{ subterm of } s, \text{ such as } t \in \mathcal{K}_i\}$
This system permits to formally define security properties related to group protocols. Some of them are strongly linked to the time evolution of the group, such as the independence of group keys, the forward secrecy or the backward secrecy. Moreover, other properties are time independent, like the implicit authentication, the secrecy, the confirmation or the integrity. The modelling of these security properties is based on the interaction of subsets defined above. For instance, the security property to be verified for the Asokan-Ginzboorg protocol is the key agreement property. It says that for the same session, the group members have to deduce the same group key. This property is specified in our model by: $\forall a_i, a_j \in \mathcal{A}\ with\ i \neq j,\ Alg_i = Alg_j$.

For our example with two participants (A_1 and A_2), the group key agreement is violated when: $Alg_{12} \neq Alg_{22}$, that is $F(x_{10}, S_{22}) \neq F(x_{13}, x_{14})$, or $Alg_{11} \neq Alg_{21}$, that is $F(x_3, S_{11}) \neq F(x_6, x_7)$. Therefore, the violation of the group key agreement property can be specified by the following constraints: $x_{10} \neq x_{13}$ or $x_{14} \neq S_{22}$ or $x_6 \neq x_3$ or $x_7 \neq S_{11}$.
For more details about the modelling and the verification of other properties, the reader must refer to the model presented in [4]. In the present paper we show how this model can be used to search for attacks in group protocols.

5 Searching for Attacks in Group Protocols

We present in this section the algorithm of searching for attacks. It is described as follows:

Algorithm *AttackSearch*(ppty,instance)
  execCorrect = True
  Exec = $\{(\emptyset, \{\mathcal{R}_p \rightarrow \mathcal{S}_p\}_{p \in \mathcal{P}}, S_0, \emptyset)\}$
  SC_P = *ConstraintsPpty*(ppty,instance)
  While execCorrect = True **and** Exec $\neq \emptyset$ **Do**
    choose (PT,PTT,S,SC)$\in$ Exec
    canCompose = True
    While canCompose = True **and** $PTT \neq \emptyset$ **do**
      choose p minimal such as $\mathcal{R}_p \rightarrow \mathcal{S}_p \in PTT$
      If *Compose*($\mathcal{R}_p$,S) **then**
        Treat($\mathcal{R}_p \rightarrow \mathcal{S}_p$,S,SC)
        Take $\mathcal{R}_p \rightarrow \mathcal{S}_p$ from PTT
        Add $\mathcal{R}_p \rightarrow \mathcal{S}_p$ to PT
      else
        canCompose = False

```
    EndIf
  End
  If canCompose = True Then
    If Attack(SC,SC_P) Then
      execCorrect = False
    EndIf
  EndIf
End
```

The algorithm takes as parameters the instance of the protocol and the property to verify.

The first step of the procedure of searching for attacks consists in the generation of the constraints set (SC_P) related to the violation of the security property given in parameter as 'ppty'. This is done in the algorithm by the function *ConstraintsPpty*. With this intention, we follow the steps described below:

- the services driven modelling of the protocol's instance (parameter 'instance');
- the services driven modelling of the violation of the property (parameter 'ppty');
- the deduction of the set of constraints related to the violation of the security property.

The constraints set generated constitutes the first level of the constraints tree (to be explained in Section 6).

The idea behind the algorithm is to consider all the possible executions of the protocol in order to find one execution corresponding to an attack. An execution is defined by the quadruple (PT,PTT,S,CS) where,

- PT: the set of steps of the execution belonging to $\{\mathcal{R}_p \rightarrow \mathcal{S}_p\}_{p\in\mathcal{P}}$ which are already treated. They are messages already exchanged between honest participants and the Intruder. At the beginning of the procedure, this set is empty.
- PTT: the set of steps of the execution belonging to $\{\mathcal{R}_p \rightarrow \mathcal{S}_p\}_{p\in\mathcal{P}}$ which have not been treated yet. This set is provided with a total order to say that a step must be treated before an other and thus a message must be exchanged before an other. Initially, this set contains all the steps given as parameter to the procedure *AttackSearch*.
- S: the set of the Intruder knowledge after the last treated step. Initially, this set is equal to the set S_0 of the instance given as parameter to the procedure *AttackSearch*.
- CS: the constraints set of the protocol. At the beginning of the procedure, this set is empty.

Consider an execution of the protocol among the (finite) set of possible executions. An execution corresponds to an attack if the constraints generated throughout all the execution's steps (denoted SC) are coherent with the constraints of the violation of the secutity property. This test of coherence can be done at the end of the execution. In fact, for each step of the execution, we consider the rule $\mathcal{R}_p \rightarrow \mathcal{S}_p$. The aim of the Intruder is to compose from his current knowledge a term corresponding to the pattern of the term $\mathcal{R}_p$. In the algorithm, this is tested by the function *Compose*. This function permits to test if the Intruder can compose the message $\mathcal{R}_p$ from his current knowledge. We note that this function uses only the Intruder composition rules since we assume that the set of the Intruder knowledge S contains only terms that cannot be decomposed yet. This hypothesis is maintained thanks to the Intruder knowledge management (to be explained in Section 7). The function *Compose* returns a boolean result. If the result is negative, then the Intruder fails in composing such a message, and so, he cannot go on in the execution. Thus, this execution cannot lead to an attack. In this case, we consider another execution.

If the result is positive, then we have to treat the current step $\mathcal{R}_p \rightarrow \mathcal{S}_p$ of the instance. This is the role of the function *Treat*. The matching between composed messages and the message

expected leads to the construction of constraints joining informations (constants or variables) given in the message expected to the ones in the composed messages. We note that, in the case where the Intruder can compose several messages matching with the expected message, we obtain several alternatives and thus several choices of constraints (presented as a disjunction). These constraints are added to the set of the protocol constraints SC.
Behind this addition, there is an important point that we must focus on: the management of constraints, especially when there is a huge constraints' set to be added only in one step of an instance. This will be discussed in Section 6.

Once the messages are composed, the Intruder gets new knowledge that he will use in the next instance's steps. Therefore, the Intruder's knowledge must be updated for each acquisition of new information. Thus, we have to manage the Intruder knowledge set S for each step of a protocol instance. Then, we add to this set S the new information acquired while taking into account the definition of S as the set where we cannot apply the Intruder decomposition rules to its terms yet. This notion will be more studied in Section 7.

At the end of the execution, the two sets of constraints SC and SC_p are solved in order to get a solution that relates variables of the execution's steps with constants or with each other. This is the aim of the function *Attack*. It tests if the two constraints' sets are coherent. The two sets are coherent if and only if there exists at least one path in the constraints tree that contains only coherent constraints when the execution finishes (see Section 6).
If the two sets are coherent then the tested security property is violated for the concerned execution. In this case, the function permits to solve the constraints based on the union of the two sets SC and SC_P. While instantiating variables in the execution in question with the solution found, we obtain the execution's trace of the attack.

This method is efficient since the search for flaws is static as it corresponds to a resolution of two constraints systems. Nevertheless, for an execution step, generating the constraints matching all possible composed messages to the expected message can lead to a huge set of constraints. In order to minimize this set, we propose two kinds of suggestions:

- In the first one, for an execution step, we consider only constraints relying on variables used in the property's violation constraints (we note V this set of variables). For the other variables, we just save the information that the message must be composed from a certain set of knowledge: the **current** Intruder knowledge.
- The second proposition is based on the combination between the construction of the constraints set of the protocol SC and the test of coherence of the two sets of constraints SC et SC_P. Indeed, while generating the constraints of the protocol related to one step, we test the coherence of this subset of constraints with all the constraints built before. This permits to eliminate unnecessary constraints. This can be done by the use of the constraints tree (see Section 6).

6 Constraints Management

Our method for searching for flaws is based on the constraints' solving. The first part of constraints comes from the modelling of the violation of the security property to be tested in the services driven model (see Section 4). The second part is generated and updated at each step treated among the different steps composing the protocol's instance. It is to be noticed that these steps are totally ordered in an execution.
Since the aim of our procedure is to search for an eventual attack, it's goal is to find an execution that corresponds to two coherent sets of constraints SC and SC_P. Knowing the set SC_P, in order

to minimize the number of constraints added in an execution's step, we just add the necessary constraints that are coherent with the ones of the previous steps. To do this, we propose to associate the execution tree to a constraints tree. The idea behind the constraints tree is to allow only the addition of the necessary constraints to the set SC and thus to consider only constraints that are coherent with the ones of the previous steps.

The constraints tree is initially constructed from the constraints related to the violation of the security property to be tested. These constraints represent the first level of the tree. We note V the set of variables given in the current constraints of the protocol. This set is initiated to the variables manipulated in the first level of the constraints tree. Besides, as constraints of the first level can explain different choices to violate the property to be verified, the level concerned (the first) is composed of different states representing these alternatives. For each of these states, we consider the possible executions with the intention to find one corresponding to an attack.

Moreover, the tree has to be updated for each step $\mathcal{R}_p \rightarrow \mathcal{S}_p$ to be treated. We assume that we are at the level i of the constraints tree. For each state of the level i, we focus on constraints corresponding to the current step and coherent with constraints related to the current state of the level i. Once the Intruder is able to compose message(s) looking like the message $\mathcal{R}_p$, we can generate different alternatives for the constraints matching this (these) message(s) to the message $\mathcal{R}_p$ expected by the honest participant. Since these constraints can be different alternatives to form the message expected, they can be eventually represented by different states at the level $i+1$. Let us note the set of the constraints representing an eventual state of the level $i+1$ as *scc*. While updating the constraints tree, we have to distinguish different cases:

- Constraints of *scc* do not contain variables that already exist in a previous level relating the root to the direct parent. In this case, we may just save the information that this variable (the message in general) has to be generated from a certain set of knowledge (the current set of the Intruder knowledge). This information would be useful whenever another lower level use the same variable.
- Constraints of *scc* contain variables that already exists in a previous level relating the root to the direct parent. In this case, there are two possibilities:
  - Constraints of *scc* are incoherent with those of parents (constraints that exist between the direct parent (current state of the level i) and the root (alternative of the violation of the security property)). In this case, we do not add this state to the $(i+1)$-th level.
  - Constraints of *scc* are coherent with these of upper parents. In this case, we maintain these constraints *scc* as a possible son of the state at the level $i+1$.
    Besides, if the constrains of this possible state contains one constraint that already exists in previous states or may be deduced by transition, we may omit it in order to get rid of redundancy.
    Moreover, if a constraint of *scc* manage variables that already exist in V (already have values) and relating them to variables that are not yet in V, this constraint is replaced by a new one connecting the new variable to its value (by transition).

7 Intruder Knowledge Management

In our method for flaws detection we assume that our Intruder follows the most referred Intruder's model: the Dolev-Yao's model [7]. In this model, the Intruder has the entire control of the communication network. That is to say that the Intruder can intercept, record, modify, compose, send, encrypt and decrypt (if he has the appropriate key) each message. He has also the possibility to send faked messages in the name of another participant. Since he has such capabilities, the Intruder has to manage information he acquires from each step. We note the set of his knowledge

S. This set is defined as the set of all present knowledge in its maximality decomposed form. From the beginning of a protocol's execution, the Intruder knows some information. Terms composing these information constitute the set of his initial knowledge S_0. Then, initially, $S = S_0$. Throughout an execution, the set of the Intruder's knowledge has to be updated for each step treated. We distinguish two kinds of updates: when the Intruder has to compose a message(s) having the same pattern as the message expected by an honest participant, and when he gets some new information (as response from an honest participant).

Operation	**Composition Rules**	**Decomposition Rules**
Fresh	$\frac{}{k}$, $k \notin \mathcal{K} \cup \mathcal{A} \cup \mathcal{S}$	
Concatenation	$\frac{m1 \quad m2}{< m1, m2 >}$	$\frac{< m1, m2 >}{m1}$, $\frac{< m1, m2 >}{m2}$
Asymmetric Encryption	$\frac{m \quad k}{\{m\}_k^p}$	$\frac{\{m\}_k^p \quad inv(k)}{m}$
Symmetric Encryption	$\frac{m \quad b}{\{m\}_b^s}$	$\frac{\{m\}_b^s \quad b}{m}$
Product	$\frac{x \quad y}{x.y}$	$\frac{x.y \quad y^{-1}}{x}$
Inverse	$\frac{y}{y^{-1}}$	$\frac{y^{-1}}{y = \{y^{-1}\}^{-1}}$
Exponentiation	$\frac{t \quad \alpha}{\alpha^t}$	$\frac{\alpha^{x.y} \quad y^{-1}}{\alpha^x}$

Table 1. Rules of the Intruder's terms composition and decomposition

In the first case, we are treating the step $\mathcal{R}_p \rightarrow \mathcal{S}_p$ where the Intruder has to build a message(s) suiting the pattern of the message expected by an honest participant ($\mathcal{R}_p$). To do this, the Intruder follows composition rules defined in the first part of the Table 1.
In the second case, since all the messages sent by the participants acting in the protocol are sent to the Intruder, the last one has the possibility to decompose the message received by using terms already existing in S at this moment. This set S can also contain terms that have not yet be decomposed because some information was missing. Therefore, from information deduced from the last message received, the Intruder can decompose terms in S. In order to decompose terms, the Intruder follows decomposition rules defined in the second part of the Table 1.
We return now to our running example. In this paragraph, we consider the following execution's order ($<_e$): $p_{111} <_e p_{212} <_e p_{211} <_e p_{112} <_e p_{121} <_e p_{222} <_e p_{221} <_e p_{122} <_e p_{131} <_e p_{232}$
The aim of the Intruder is to build the patterns of the messages expected by the honest participants. We notice that to fulfill this goal, we need to manage the Intruder's knowledge whenever he gets a new information (that comes from a message received). For each message expected (taking into account the execution's order), the Intruder build every message that looks like the pattern of the message in question. By application of the Flaws Detection method to our example of the Asokan-Ginzboorg protocol, we find the constraints expressed in Table 2.

While solving the constraints system listed in Table 2 with the one of Section 4 (the one of the security property to verify) we find this solution: $x_{10} = x_6 = S_{21},\ x_2 = R_2,\ x_7 = S_{22},\ x_{12} = E_1,\ x_{13} = x_3 = S_{12},\ x_5 = E_2,\ x_9 = R_1$and $x_{14} = S_{11}$.
The instantiation of variables in the execution by the values found above gives us the execution's trace of Figure 1.

At the end of this execution's trace, we have: $Alg_{11} \neq Alg_{21}$ and $Alg_{12} \neq Alg_{22}$. Thus, the group key agrement is violated for each one of the two sessions.

1	$x_5 = E_1$ *or* $x_5 = E_2$
2	$x_{12} = E_1$ *or* $x_{12} = E_2$
3	$x_2 = R_1, x_3 = S_{21}, E_1 = x_5$ *or* $x_2 = R_2, x_3 = S_{12}, E_1 = x_{12}$
4	$x_9 = R_1, x_{10} = S_{21}, E_2 = x_5$ *or* $x_9 = R_2, x_{10} = S_{12}, E_2 = x_{12}$
5	$x_6 = x_3, x_7 = S_{11}, x_2 = R_1$ *or* $x_6 = x_{10}, x_7 = S_{22}, x_9 = R_1$
6	$x_{13} = x_3, x_{14} = S_{11}, x_2 = R_2$ *or* $x_{13} = x_{10}, x_{14} = S_{22}, x_9 = R_2$
7	$x_3 = S_{21}, x_6 = S_{21}, x_7 = S_{11}$ *or* $x_3 = x_{13}, x_3 = S_{12}, x_{14} = S_{11}$
8	$x_{10} = S_{21}, x_6 = S_{21}, x_7 = S_{22}$ *or* $x_{10} = S_{12}, x_{13} = S_{12}, x_{14} = S_{22}$

Table 2. Constraints for the Asokan-Ginzboorg protocol

$$
\begin{array}{lrll}
\mathbf{p_{111}}: & i & \longrightarrow A_1 : Init & \\
 & A_1 & \longrightarrow i \quad A_1, \{E_1\}_P & \\
\mathbf{p_{212}}: & i & \longrightarrow A_2 : Init & \\
 & A_2 & \longrightarrow i \quad A_2, \{E_2\}_P & \\
\mathbf{p_{211}}: & i & \longrightarrow A_2 : x_4, \{E_2\}_P & \\
 & A_2 & \longrightarrow i \quad A_2, \{R_1, S_{21}\}_{E_2} & \\
\mathbf{p_{112}}: & i & \longrightarrow A_1 : x_{11}, \{E_1\}_P & \\
 & A_1 & \longrightarrow i \quad A_1, \{R_2, S_{12}\}_{E_1} & \\
\mathbf{p_{121}}: & i & \longrightarrow A_1 : x_1, \{R_2, S_{12}\}_{E_1} & \\
 & A_1 & \longrightarrow i \quad \{S_{12}, S_{11}\}_{R_2} & \\
\mathbf{p_{222}}: & i & \longrightarrow A_2 : x_8, \{R_1, S_{21}\}_{E_2} & \\
 & A_2 & \longrightarrow i \quad \{S_{21}, S_{22}\}_{R_1} & \\
\mathbf{p_{121}}: & i & \longrightarrow A_2 : \{S_{21}, S_{22}\}_{R_1} & \\
 & A_2 & \longrightarrow i \quad A_2, \{S_{21}, H(S_{21}, S_{22})\}_{F(S_{21}, S_{22})}, & \mathbf{Alg_{21}} = \mathbf{F(S_{21}, S_{22})} \\
\mathbf{p_{122}}: & i & \longrightarrow A_1 : \{S_{12}, S_{11}\}_{R_2} & \\
 & A_1 & \longrightarrow i \quad A_1, \{S_{12}, H(S_{12}, S_{11})\}_{F(S_{12}, S_{11})}, & \mathbf{Alg_{22}} = \mathbf{F(S_{12}, S_{11})} \\
\mathbf{p_{131}}: & i & \longrightarrow A_1 : x_1, \{S_{12}, H(S_{12}, S_{11})\}_{F(S_{12}, S_{11})}, & \mathbf{Alg_{11}} = \mathbf{F(S_{12}, S_{11})} \\
\mathbf{p_{232}}: & i & \longrightarrow A_2 : x_8, \{S_{21}, H(S_{21}, S_{22})\}_{F(S_{21}, S_{22})}, & \mathbf{Alg_{12}} = \mathbf{F(S_{21}, S_{22})}
\end{array}
$$

Fig. 1. Execution's trace

8 Verification Results

By applying the strategy described in previous sections, we have found two authentication attacks for the protocol GDH.2 with 4 participants (attacking resp. participant A_3 and A_4) (See Figures 2 and 3). These Figures show either the normal execution of the protocol and the message to be changed in order to lead to an attack. In this section, we generalize these results to the GDH.2 with n participants and to the protocol A-GDH.2.

8.1 The GDH.2 protocol

Consider the GDH.2 [9] protocol with n participants $(A_1, ..., A_n)$. The Intruder can have the point of vue of the group key of the last member. Indeed, he intercepts the last message intended for the last participant $(x_1, ..., x_{n-1}, x_n)$ and alters the last component (x_n) by replacing it by any component of the last message varying from x_1 to x_{n-1}. When receiving the message expected, the last participant A_n exponentiates the components $x_1...x_{n-1}$ by R_n and send them to the other participants. The Intruder can then get the information varying from $Exp(x_1, R_n)$ to $Exp(x_{n-1}, R_n)$.

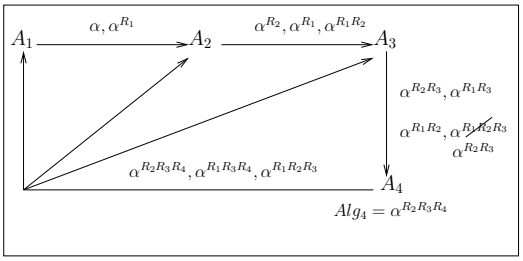

Fig. 2. First authentication attack for GDH.2

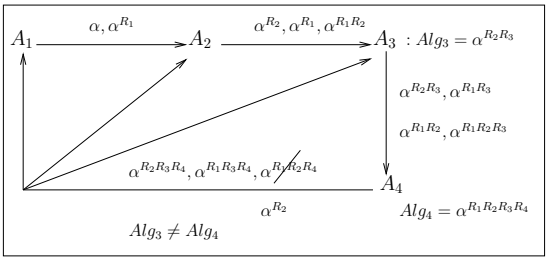

Fig. 3. Second authentication attack for GDH.2

Then, A_n uses the last component x_n in order to deduce the group key by exponentiating this by R_n. This key from the point of vue of the last member is then $Alg_n = Exp(x_n, R_n)$. Thus, if the Intruder replaces x_n by a message x_i from x_1 to x_{n-1}, the last participant deduces as group key $Exp(x_i, R_n)$. Nevertheless, this information is already available on the network and then known by the intruder.

The aim of the Intruder is now to have the point of vue of the group key of an intermediate participant A_i varying from A_1 to A_{n-1}. A_i deduces his key from the last message he received (X) from A_n where $X = x_1, .., x_{n-1}$. Since his group key will contain a private information: R_i, the Intruder has to make so that A_i generates for key of group a message which was transmitted before and which contains private information R_i. These messages are accessible to the intruder at the time of the first round of the protocol: when the members (in particular, intermediate members) are invited to give their contributions by exponentiating the messages received by the R_i. At the step p_i, A_i receives a message of form $x_{11}, .., x_{1i}$. In the message to be sent by A_i during this step, we find all the components x_{1j} (varying from x_{11} to x_{1i}) exponentiated by R_i. We assume now that, for an $i \in \{1, n-1\}$, for the message $X = x_1, .., x_i, ..., x_{n-1}$, x_i is one of the components $x_{11}, .., x_{1i}$. The participant A_i, while receiving X, takes his correspondent component x_i and generates his key by exponentiating it by R_i. Thus, $Alg_i = Exp(x_i, R_i)$. Nevertheless, the Intruder got already the information $Exp(x_i, R_i)$ from the step p_i.

8.2 The A-GDH.2 protocol

Consider the A-GDH.2 [9] protocol with n participants $(A_1, ..., A_n)$ where the Intruder is one of the participants and has as raw i $(I = A_i)$. The Intruder focus on the last message expected by the last participant. This message X is composed of n components $X = x_1...x_n$. The last participant exponentiates the (n-1) components of this message by R_n and the key of the corespondent participant. Thus, the component x_i is exponentiated by R_nK_{ni}. Instead of sending the message $X = x_1, ..., x_i, ..., x_n$ to A_n, the Intruder send the message $X = x_1, ..., x_i, ..., x_i$. As response to this message, A_n sends the message
X' = $Exp(Exp(x_1, R_n), K_{n1}), ..., Exp(Exp(x_i, R_n), K_{ni}), ..., Exp(Exp(x_{n-1}, R_n), K_{n(n-1)})$
and deduces as group key $Exp(x_i, R_n)$ as $x_n = x_i$. From the message X', the Intruder gets the component $Exp(Exp(x_i, R_n)K_{ni})$. He knows then $Exp(x_i, R_n) = Exp(x_n, R_n)$. Thus, he has the point of vue of the group key of the participant A_n. Moreover, for the other group members, as the Intruder does not alter the rest of the normal protocol's execution, he shares the same expected group key with the rest of the group (apart from A_n) $Exp(Exp(x_1, R_n), R_1) = Exp(Exp(x_i, R_n), R_i) = Exp(Exp(x_{n-1}, R_n), R_{n-1})$. The Intruder succeed then to divide the group on two parts and has the two points of view of the group key of the two parties.

9 Related Work

The verification of group protocols is a research topic on which there has been and there is still a lot of work. This is mainly due to the wide range of specific requirements imposed by this kind of protocols. Varying from an unbounded number of participants to very particular security properties, considering all those requirements is a real challenge, both theoretical and practical. Either modelling and verifying group protocols and their properties are very difficult.

It exists various research activities to formally specify group protocols and their specific requirements. For instance, Capsl has been extended to MuCapsl [6]. This language is to be translated to a multiset term rewriting rules (MuCIL) which is an extension of CIL in order to support multicast group management protocols. However, we only model the security property of secrecy.
In a recent work [5], Delicata and Schneider present a framework for reasoning about secrecy in a class of Diffie-Hellman protocols. The technique, which shares a conceptual origin with the idea of a rank function, uses the notion of a message-template to determine whether a given value is generable by an intruder in a protocol model. This work focus only on the security protocol of secrecy. Then, it is less general than Pereira's work as it deals with a sub class of Diffie-Hellman protocols (it claims the condition of I/O independence).

In the verification process, after the first step: the specification of either the protocol and the property, there is a more delicate step which is the verification step. It exists various research activities oriented on this task varying from manual methods to automatic ones. They lead to the discovery of several attacks that will be introduced in this section.

One of the most interesting techniques done by hand is suggested by Pereira and Quisquater in [9]. They have introduced a method converting the problem of ownership of some information by the intruder to a problem of resolution of a system of linear equations. With this method, several attacks have been found in the protocols suite CLIQUES [9]. This method has also permitted to get a generic result: it is impossible to design an authentication group key agreement protocol built on A-GDH for a number of participants greater than or equal to four [10]. Although this method is of great interest for analysing group protocols, its main drawback is that it has to be run by hand for discovering attacks.

Additionally, some tools have been extended in order to deal with the new requirements of group protocols. Significant attacks on such protocols have been found. In [12], Taghdiri and Jackson have modelled a multicast group key management protocol proposed by Tanaka and Sato [13]. They have been able to discover counterexamples to supposed properties. They have then proposed an improved protocol. However, in their model, no active attacker was included. Their improved protocol has been analyzed in [11] by CORAL and two serious attacks have been found. CORAL has also been used to discover other attacks concerning two protocols: Asokan-Ginzboorg and Iolus.

10 Summary and future work

Throughout this paper, we have presented a new strategy for dealing with group protocols and more generally contributed ones. The approach hinges around the use of the services driven model to deduce constraints related to the security property to verify. These constraints will be used with the protocol's execution constraints to obtain an attack execution's trace if it exists. This strategy permits to pinpoint new attacks in three different protocols. From these attacks, we have generalized the result to two protocols with n participants.

This work is nascent, but we are currently applying it to other protocols and to other security properties. We also plan to study the complexity of the suggested algorithm.

Since the analysis of a great number of protocols is generally done by automatic tools, we intend either to implement our strategy or to extend existing automatic tools that are based on constraints solving. Among these tools, we find Atse, one of four back-ends used in AVISPA [1], a tool that has already treated a large number of Internet security protocols. Its expressive protocol specification language permits, modulo some extensions, to model contributed protocols and their intended security properties. Since our basic constraints are based on equality and inequality constraints, they may be seen as booleen constraints and then the whole constraints can be considered as SAT constraints. Thus, we may integrate a SAT-solver in our solution.
The suggested approach can also be developed to consider another kind of group protocols such as hierarchical protocols that present additional verification constraints. Indeed, we have to extend the services driven model to deal with this kind of protocols.

References

1. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santos Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, Edinburgh, Scotland, 2005. Springer.
2. N. Asokan and P. Ginzboorg. Key Agreement in ad hoc Networks. *Computer Communications*, 23(17):1627–1637, 2000.
3. Y. Chevalier and L. Vigneron. Strategy for Verifying Security Protocols with Unbounded Message Size. *Journal of Automated Software Engineering*, 11(2):141–166, 4 2004.
4. N. Chridi and L. Vigneron. Modélisation des propriétés de sécurité de protocoles de groupe. In *Actes du 1 er Colloque sur les Risques et la Sécurité d'Internet et des Systèmes*, pages 119–132, Bourges, France, October 2005. CRISIS.
5. R. Delicata and S. Schneider. A formal approach for reasoning about a class of diffie-hellman protocols. In *Formal Aspects in Security and Trust*, pages 34–46, 2005.
6. G. Denker and J. Millen. Modeling group communication protocols using multiset term rewriting, 2002.
7. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.
8. H. Hassan, A. Bouabdallah, H. Bettahar, and Y. Challal. Hi-kd: Hash-based hierarchical key distribution for group communication - ieee infocom poster, 2005.
9. O. Pereira. *Modelling and Security Analysis of Authenticated Group Key Agreement Protocols*. PhD thesis, Universit catholique de Louvain, May 2003.
10. O. Pereira and J.-J. Quisquater. Generic Insecurity of Cliques-Type Authenticated Group Key Agreement Protocols. In *17th IEEE Computer Security Foundation Workshop, CSFW*, pages 16–19, Pacific Grove, CA, 2004. IEEE Computer Society.
11. G. Steel and A. Bundy. Attacking group multicast key management protocols using coral. *Electr. Notes Theor. Comput. Sci.*, 125(1):125–144, 2005.
12. M. Taghdiri and D. Jackson. A Lightweight Formal Analysis of a Multicast Key Management Scheme. In *Formal Techniques for Networked and Distributed Systems, FORTE*, volume 2767, pages 240–256, Berlin, Germany, 2003. Springer.
13. S. Tanaka and F. Sato. A Key Distribution and Rekeying Framework with Totally Ordered Multicast Protocols. In *15thon Information Networking, ICOIN*, pages 831–838, Beppu City, Japan, 2001. IEEE Computer Society.
14. C. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. In *Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 68–79, 1998.