Rank Theorems for Forward Secrecy in Group Key Management Protocols

Amjad Gawanmeh and Sofiène Tahar
Department of Electrical and Computer Engineering
Concordia University
1455 de Maisonneuve West,
Montreal, Quebec H3G 1M8
{amjad,tahar}@ece.concordia.ca

December, 2006

Abstract

Design and verification of cryptographic protocols has been under investigation for quite sometime. However, not much attention has been paid for the class of protocols that deals with group key management and distribution, mainly because of their dynamic characteristics. In addition, these protocols have special security properties, such as forward secrecy, that cannot be verified using methodologies designed for normal two-parties protocols. In this paper, we provide a set of generic formal specification requirements for group key management and distribution protocols. This can help guiding the proper specification of the behavior of such protocols, which is necessary for a successful design and verification process. We define a formal model for the protocol and establish rank theorem for forward properties based on the above requirements. Rank theorems imply the validity of the security property to be proved, and are deducted from a set of rank functions we define over the protocol. The above formalizations and rank theorems were implemented using the PVS theorem prover. We illustrate our approach on the verification of forward secrecy for the Enclaves protocol designed at SRI.

1 Introduction

Cryptographic protocols provide security services for communicating entities. They involve a precise interaction between the entities in order to achieve the required security services, therefore, it is very important to verify that the protocol operations are not vulnerable to attacks.

There are different kinds of environments that protocols must interoperate with. Besides, networks handle more and more tasks in a potentially hostile environment. Therefore, cryptographic protocols should take more responsibilities in order to capture these new requirements. Some security properties like availability and fairness take more important roles in some protocols like in commercial systems. This requires that the complexity of the cryptographic protocol should be increased. In addition, new cryptographic primitives are being adopted; for instance, in group key management protocols of unbounded size, it is not obvious how to reason about them with existing protocol analysis systems. This of course, makes both verification and implementation more difficult. It also requires the search for new modeling and verification approaches for cryptographic protocols [10].

Distributing the group key to legitimate members is a complex problem. Although re-keying a group, by sending the new group key to the old group members encrypted with the old group key is trivial, re-keying the group after a member leaves is far more complicated. The old key cannot be used to distribute a new one, because the leaving member knows the old key. Therefore, a group key distributor must provide another scalable mechanism to distribute keys to the group. So, there should be a mechanism to create secret keys and distribute them among legitimate principals that guarantee the secrecy of these keys. Even though such protocols claim forward and backward secrecy, the formal analysis and verification of these properties have received very little attention.

The general requirements for protocols involving two or three parties are well understood, however, the case is different with group key distribution protocols, where the key can be distributed among a larger number of members who may join or leave the group at arbitrary times. Therefore, security properties that are well defined in normal two-party protocols have different meanings and different interpretations in group key distribution protocols, and so they require a more precise definition before we look at how to verify them. An example of such properties is the secrecy property, which deals with the fact that secret data should remain secret and not compromised. However, for group key distribution protocols, this property has a further dimension since there are long-term secret keys, short-term secret keys, in addition to present, future, and past keys; where a principal who just joined the group and learned the present key should not be able to have enough information to deduce any previous keys, or similarly a principal who just left the group should not have enough information to deduct any future keys. Therefore, systems designed for two-party protocols may not be able to model a group protocol, or its intended security properties because such tools require an abstraction to a group of fixed size to be made before the automated analysis takes place. This can eliminate chances of finding attacks on the protocol.

A limited effort has been done on modeling and verifying protocols that involve more than two parties. In addition, there are very few trials in the open literature that discuss the general formal requirements for reasoning about cryptographic protocols, which, once developed, can be applied in the design process of new protocols. In this paper, we propose a verification methodology for group protocols, in which we first give a formal requirements specification for group key distribution protocols and then discuss the verification steps based on these requirements and dedicated rank theorems for security properties.

Our verification methodology is based on the notion of rank theorems we present on this paper utilizing the *rank functions* first proposed Ryan and Schneider [11]. We map the requirements into ranks, this map is based on a predefined function, *the rank function*. For this map, we have to find the appropriate rank functions for the protocol events, traces and properties. This rank function is tailored for the security property we intend to verify, forward secrecy and backward secrecy in our case. Based on the above rank functions, we define in this paper a set of rank theorems for forward secrecy.

The proof establishment will be mechanized in the PVS (Prototype Verification System) theorem prover [8].Rank theorems, protocol events and traces of executing have been embedded in PVS. We apply the implemented proof environment on the Enclaves protocol from SRI [3] in order to verify related forward secrecy property, we believe that verifying backward secrecy property will be similar.

In our approach, we can consider establishing the proof at two levels of abstraction of the protocol: the protocol level and the encryption level. At the protocol level, embedding of the rank functions and rank theorems in PVS will make the verification feasible, however, working out the proof at the encryption level, will require the definition of probabilistic theorems in PVS which are not available yet.

The rest of the paper is organized as follows, Section 2 provides related work to ours. In Section 3, we present the overall verification methodology. In Section 4, we provide our formal specification requirements model for group key management protocols, and use the definition of rank functions to establish rank theorems, then prove the theorem for forward secrecy property. In Section 5, we describe the details of our implementation of the formal specifications and theorems in PVS. Section 6 illustrates our approach by verifying the forward secrecy for the case study of the Enclaves protocol. Finally, Section 7 concludes the paper with future work hints.

2 Related Work

In this section we discuss approaches for modeling and verification of group key management and distribution protocols that are closely related to our work.

Syverson and Meadows [14] presented the formal requirements for authentication in key distribution protocols. They tried to provide a single set of requirements to specify a whole class of protocols, which can be fine-tuned for the particular application. There were two main problems in their approach; first, the requirements they provide was for a single property, authentication, which is similar in different protocols, whereas other properties may have different semantics in different classes of protocols; like secrecy property for example. Second, the requirements are defined as a security property, whereas the definition should include the whole protocol requirements and how they can be interpreted and then applied into a specific protocol.

Layouni *et al.* [5] used a combination of model checking in order to verify authentication property, theorem proving to verify safety and liveness properties such as proper agreement, and a Random Oracle Model to manually prove robustness and unpredictability properties. This example shows how difficult it is to verify and analyze this class of protocols. While the authors achieved a promising success in verifying a complex protocol such as Enclaves, they failed to achieve A formal proof of the three components in a single formalism. The authors, however, suggest that this work can be complemented by performing the analysis of the group key management module in PVS in order to be able to verify properties such as forward and backward secrecy.

Meadows and Syverson [6] used the NPATRL language, a temporal requirement specification language for use with the NRL Protocol Analyzer, in order to specify the Group Domain of Interpretation (GDOI) key management protocol. In a later stage Meadows *et al.* [7] gave a detailed specification of the requirements for GDOI and provided a formal analysis of the protocol with respect to these requirements using the NRL Protocol Analyzer. However, the problem with this approach is that there is no general set of requirements for protocols requirements which can be applied on a specific protocol, or can be used for the refinement of protocol specifications during the design process. In a related approach, Denker and Millen [2] used multiset term rewriting in order to model group communication protocols. They show the mechanisms used in key distribution and provides an analysis of group protocols complexity in terms of key distribution. This latter is useful in our case to formally define forward and backward secrecy properties, since they show in their analysis how distributing new keys may affect previously used ones. Archer [1] provided a mechanized correctness proof of the basic TESLA protocol based on establishing a sequence of invariants for the protocol using the tool TAME. The model of the protocol is rather simple, and the proof was made under a strong assumption stating that the adversary has no initial knowledge, and can only use facts revealed by users.

In a more recent work, Pereira [9] proposed a systematic approach to analyze protocol suites extending the Diffie-Hellman key-exchange scheme to a group setting. He pointed out several unpublished attacks against the main security properties claimed in the definition of these protocols. The method provided is essentially manual and applicable only on Group Diffie-Hellman (GDH) protocols. In a similar work, Sun and Lin [13] extended the strand space theory to analyze the dynamic security of Group Key Agreement Protocols (GKAP) and discussed the conditions of the security retention in the dynamic cases of the protocol, this work treats the analysis dynamic aspects of the protocol with no reasoning about the correctness of the protocol under these dynamic events. A related work by Steel *et al.* [12] model a group key protocol by posing inductive conjectures about the trace of messages exchanged in order to investigate novel properties of the protocol, such as tolerance to disruption, and whether it results in an agreement on a single key. The method, however, is applicable on limited groups of two or three members only. Recently, Truderung [15] presents a formalism, called selecting theories, which extends the standard non-recursive term rewriting model and allows participants to compare and store arbitrary messages. This formalism can model recursive protocols, where participants, in each protocol step, are able to send a number of messages unbounded w.r.t. the size of the protocol. This modeling cannot be applied on non–recursive protocols such as GDH or the Enclaves.

Ryan and Schneider [11] proposed the idea of rank functions for verification of CSP (Communication Sequential Process). Dutertre and Schneider [4] used an embedding of CSP in PVS in order to verify the authentication property of Needham Shroeder public key protocol. However, the work in [11] did not present a method that can be applied on security properties in other classes of protocols, like group key protocols. In fact, the method, as is, may not be applied on properties such as forward and backward secrecy. Even thought rank functions was first introduced and used by Ryan and Schneider, in this paper, we started from there in order to precisely define a set of requirements for rank functions and then reason about their soundness, and then suggest the new idea of using rank theorems in order to extend the use of rank functions in order to verify forward and backward secrecy properties for group key distribution protocols.

From the above account on related work, we noticed the lack of a single formalism to model the protocols and reason about their security properties, such that the protocol can fit and its verification is feasible. There is no formal link between the informal specification and the provided protocol models and their security properties. All approaches concentrate on the trivial secrecy and authentication properties. Besides, there are no trials to reason about complex features of key distribution properties such as key hierarchies that are not easy to handle. Also, there is no generalized verification methodology that can be instantiated to prove the correctness of a specific protocol. Finally, there are no well defined specification requirements, which will reduce the possibility of introducing errors into the protocol during the design process. This justifies the need for a generic set of formal specification requirements of group key distribution protocols, which is discussed in the next section.

3 Verification Methodology

Our verification methodology is based on rank theorems. We use a rank function to map facts about the protocol into ranks, and define for every security property a theorem that implies the validly of the property with respect to the protocol. In following, we briefly present the steps of our verification methodology. Figure 1 provides a summary of these steps. The first step consists of providing a formal model and precise definition for group protocols properties and events. This will help eliminating the gap between the informal protocol specification and the formal model. It will also provide a well defined protocol specification that can be directly integrated into the verification methodology. In the second step, we define map functions between the set of facts and the set of integers. The set of facts include protocol events, protocol execution traces and the security property. This mapping function will be useful in partitioning the message space and enabling mechanized proof of security protocols properties. The main idea is to define rank theorems that provide conditions satisfied by a given rank function in order to conclude that the security property satisfies its protocol model, we define for every security property a theorem that implies the validly of the property with respect to the protocol, we show the proof of the correctness of the rank theorem. The set of events and traces are concretely defined by the protocol. This allows their definition at different levels of abstraction in the final step of our approach, which is implementing the rank theorems in PVS and establishing their proof of correctness.

After this map, we define *rank theorems*, which are the set of properties and protocol specifications modeled using the rank functions we defined. Rank theorems imply the correctness of the security property it models. In order to prove the correctness of a specific property, we need to prove that its corresponding

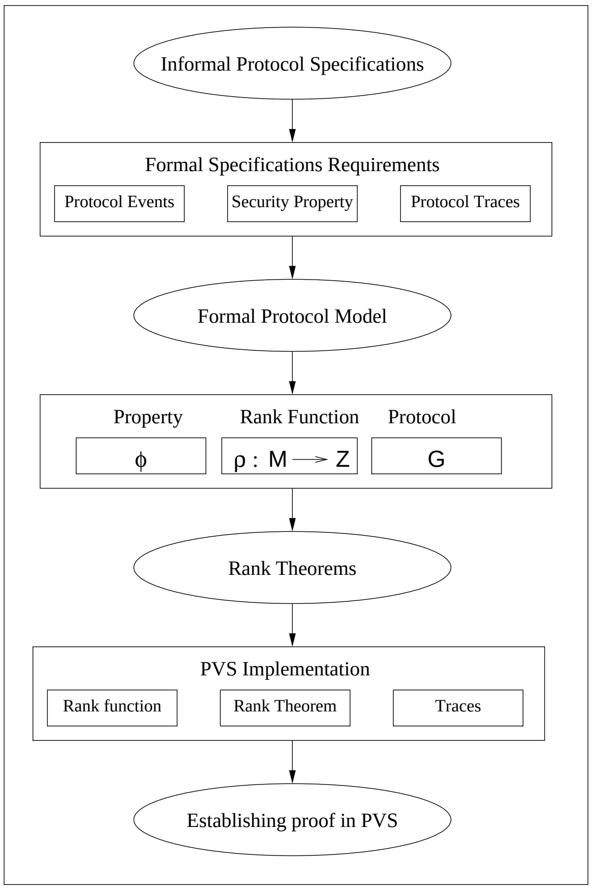

Figure 1: Verification Methodology

rank theorem is correct with respect to the protocol model. The final step is to mechanize the proof through PVS theorem prover. For this purpose, we provide PVS implementation for the rank theorem, events and traces of the protocol, and then use PVS theorem proving strategies to construct the correctness of the rank theorem.

In summary, the proposed methodology is a top-down approach, where the property verification is achieved in the last step. In order to make sure of the soundness of our approach, we have to show the formal link between the constructed rank theorem and the formal model of the property. This is carried out by proving that the correctness of the rank theorem implies the correctness of the security property. This way, we can argue that verifying the property at the implementation level guarantees the correctness of the property in the model.

4 Formal Specification Requirements Model

In this section we give the specification requirement of group key management protocols. Since there are many different approaches in the literature to design such protocols, specially keys generation and distribution, the specification of these protocols and their properties are informal. So we try to provide a common formal model for these specifications where most of commonly designed protocols fit. We need these formal specification requirements for many reasons: first, to fill the gap between the informal protocols descriptions on one hand and the formal protocol models and their implementations on the other hand. Second, to integrate formal analysis in the design process of cryptographic protocols and specifically group key distribution protocols [6]. Finally, to give a better understanding of the verification problem and suggest a verification method based on these requirements.

Freshness requirement imposes that when a principal receives a piece of information, such as keys, then this information must have been fresh and currently valid. In this sense, freshness is similar to those that have been defined for two parties protocols. *Group secrecy* should guarantee that it is computationally infeasible to discover any group key. *Forward secrecy* should guarantee that knowing a subset of old group keys will not lead to the computation of any subsequent group key. *Backward secrecy* should guarantee that knowing a contiguous subset of group keys will not lead to the computation of a preceding group key.

Group Joining or *leaving* events are operations that result in creating a new group with a new group key out of an existing group. Protocols should guarantee that above properties remain valid when members join or leave the group. A *Group Merge* group event is the operation where subgroups need to be merged back into a single group, a new group key is computed and distributed to every member of the new group, and group keys for subgroups are considered old or preceding keys. A *Group Split* group event is the operation of creating two subgroups out of a single group, where every subgroup has an independent group key.

In the rest of this section and throughout this paper, we will use the following notations:
$\mathbb{M}$: messages space, the set of all possible messages.
P: a honest principal who is willing to communicate.
$\mathbb{P}$: the set of knowledge of member P, $\mathbb{P} \subseteq \mathbb{M}$.
$\mathbb{S}$: secret messages space, the set of all secret messages, $\mathbb{S} \subset \mathbb{M}$. These are the messages we want to keep hidden from the intruder. These messages are defined by the protocol.
I: a dishonest member. We assume that the intruder is a dishonest member who is trying to find an attack in the protocol by using his unlimited resources and computational power. However, we state normal assumptions about the intruder such as being able to encrypt or decrypt a message only if he knows the appropriate key, the ability to block or read any message in the system.
$\mathbb{E}$: the set of all events, or dynamic operations, i.e., join, leave, merge, and split. An even is a term from the message space to the message space, $\mathbb{E} : \mathbb{M}^r \rightarrow \mathbb{M}^n$. It represent an action the user can perform in order to obtain extra information and update his own set of knowledge.

$\mathbb{T}$: the set of all possible traces, where a trace of events is the execution of the sequence of these events. We use $\tau \in \mathbb{T}$, such that $\tau : \mathbb{E}^n \times \mathbb{M}^p \to \mathbb{M}^r$, $m \in \mathbb{M}$, E^n is a vector of n events of type $\mathbb{E}$, and M^p is a vector of p messages of type $\mathbb{M}$, then we write $m = \tau(E^n, M^p)$ to say that a message m is generated by the trace τ by executing the vector of events E^n on the vector of messages M^p. We also write $\tau(E^n, M^p) \rightsquigarrow m$ to represent a predict formula that evaluates to true if and only if $m = \tau(E^n, M^p)$.
$\mathbb{K}_0$: the set of initial knowledge of the intruder, where $\mathbb{K}_0 \subset \mathbb{M}$. The initial knowledge of the intruder is basically the information he/she can collect before start executing the protocol events. This information is usually public and known, so there are no secret information that is in the intruders initial set of knowledge. In other words $\forall m \in \mathbb{M} : m \in \mathbb{S} \Rightarrow m \notin \mathbb{K}_0$
$\mathbb{K}$: the set of knowledge of the intruder, the intruder updates this knowledge by executing events starting with the initial set of knowledge. $\mathbb{K}_0 \subseteq \mathbb{K}$ and $\mathbb{K} \subseteq \mathbb{M}$.
$\mathbb{G}_t$: current group, which can be formally defined as a set of principals who share a secret key, or information that can be used to calculate the secret key.
$\mathbb{G}_{t+i}$: a group that can share a secret key at future time.
$\mathbb{G}_{t-i}$: a group that previously in time shared a secret key.
$\in$ group membership: we define membership as follows: $K_{\mathbb{G}_t} \in \mathbb{P} \Longrightarrow P \in \mathbb{G}_t$, which means a principal P is a member of the group $\mathbb{G}_t$ at this time, t, if the group key $K_{\mathbb{G}_t}$ is in his set of principal P's knowledge $\mathbb{P}$.
$K_{\mathbb{G}_t}$: the group session key: the key generated for the current session. Equivalently, it can be the set of information that can be used to calculate the key. $I \notin \mathbb{G}_t \Rightarrow K_{\mathbb{G}_t} \in \mathbb{S}$
$K_{\mathbb{G}_{t+i}}$: a group session key for the group $\mathbb{G}_{t+i}$ that can be generated and used sometime in the future, $I \notin \mathbb{G}_{t+i} \Rightarrow K_{\mathbb{G}_{t+i}} \in \mathbb{S}$.
$K_{\mathbb{G}_{t-i}}$: a group session key that was generated and used previously in time. $I \notin \mathbb{G}_{t-i} \Rightarrow K_{\mathbb{G}_{t-i}} \in \mathbb{S}$.

4.1 Secrecy

Only members of the group should have access to keys. The important issue here, is wether we want to allow users who just joined the group to have access to previously used keys, also wether we want to allow users who just left the group to have access to keys that will be generated henceafter. To ensure the secrecy of old and new keys, every protocol uses a mechanism for keys generation to guarantee that they cannot be calculated using the current group session information including the key itself.

In the following, we give the formal definition of group secrecy, forward secrecy and backward secrecy.

Definition 4.1. *Group key secrecy: for any current group $\mathbb{G}_t$, and a dishonest principal I who knows a set of initial knowledge $\mathbb{K}_0$, there is no trace $t \in \mathbb{T}$ that he/she can execute in order to obtain the current group session key $K_{\mathbb{G}_t}$.*

$I \notin \mathbb{G}_t \Rightarrow \neg\exists\tau \in \mathbb{T}$: $K_{\mathbb{G}_t} = \tau(E^n, M^p)$, where E^n is a vector of n events of type $\mathbb{E}$ and M^p is a vector of p messages of type $\mathbb{M}$.

Forward secrecy requires that a session key cannot be calculated from keys and information that are generated before this key in time. Which means that compromising sessions keys does not compromise previous session keys that were established for previous protocol runs. In order for a protocol to satisfy this property, there should be no trace of events that can lead to generating previously used keys by a user who was not part of the group at the time when the key was generated. We formally model the forward secrecy requirement as follows:

Definition 4.2. *Forward secrecy: for any current group $\mathbb{G}_t$, and a dishonest principal I, where $I \in \mathbb{G}_t$ (I knows $K_{\mathbb{G}_t}$), there is no trace $\mathbb{T}$ that he/she can execute in order to obtain a previous group session key $K_{\mathbb{G}_{t-i}}$, where $0 < i < t$.*
$I \in \mathbb{G}_t \Rightarrow \neg\exists\tau \in \mathbb{T} : K_{\mathbb{G}_{t-i}} = \tau(E^n, M^p)$, where $I \notin \mathbb{G}_{t-i}$, and $0 < i < t$, E^n is a vector of n events of type $\mathbb{E}$, and M^p is a vector of p messages of type $\mathbb{M}$.

Backward secrecy requires that a session key cannot be calculated from keys and information that are generated after this key in time. Which means that compromising sessions keys does not compromise keys for future sessions. The main concern here is that a user, who decides to leave the group, should not be able to use the information he/she learned in order to calculate keys that may be used after he/she left. We formally model the backward secrecy requirement as follows:

Definition 4.3. *Backward secrecy: for any current group $\mathbb{G}_t$, and a dishonest principal I, where $I \in \mathbb{G}_t$ (I knows $K_{\mathbb{G}_t}$), there is no trace $\mathbb{T}$ that he/she can execute in order to obtain a previous group session key $K_{\mathbb{G}_{t+i}}$, where $i > 0$.*
$I \in \mathbb{G}_t \Rightarrow \neg\exists\tau \in \mathbb{T} : K_{\mathbb{G}_{t+i}} = \tau(E^n, M^p)$, where $I \notin \mathbb{G}_{t+i}$ and $i > 0$

4.2 Joining and Leaving Groups

Any group key distribution protocol must handle adjustments to group secrets subsequent to all membership change operations. Single member operations include member *join* or *leave*. Leave occurs when a member wants to leave the group or forced to. Join occurs when a member wants to have access to the current group. Although protocols may impose an agreement criteria on joining and leaving groups, the effect of executing the event should result in a new group setting, in case the event is executed successfully, i.e., the member is granted access to the group, or released from it.

Definition 4.4. *A principal P joins the group $\mathbb{G}_t$ if $P \notin \mathbb{G}_t$, and there exists a trace $\tau \in \mathbb{T}$ that P can execute, where $K_{\mathbb{G}_{t+i}} = \tau(E^n, M^p)$ such that $K_{\mathbb{G}_{t+i}} \in \mathbb{P}$ (or $P \in \mathbb{G}_{t+i}$) and $K_{\mathbb{G}_{t+i}} \neq K_{\mathbb{G}_n}$.*

For this definition of *join* event, there is a time delay of i, which should be less than the maximum join delay imposed by the protocol.

Definition 4.5. *A principal P leaves the group $\mathbb{G}_t$ if $P \in \mathbb{G}_t$, and there exists a trace $\tau \in \mathbb{T}$ that P can execute such that $K_{\mathbb{G}_{t+i}} \notin \mathbb{P}$ (or $P \notin \mathbb{G}_{t+i}$).*

4.3 Merging and Splitting Groups

Merging and splitting groups are considered as multiple members operations. Some protocols rely on distributing security management among distributed servers rather than on one single server. This is obtained by having multiple groups. However, sometimes there is a need to merge two groups (or more) or to split a current group into two groups. These events affect the current groups settings and result in new settings that should maintain all the security requirements of the protocol.

A merge event occurs when two groups with two different settings execute a trace of events that result in a new group setting, where every member of each of the two groups is a member of a new group. Whereas a split event occurs when one group executes a trace of events that result in two new different groups, where every member of the current group is a member of one and only one of the new groups.

After these events are executed successfully, groups operate normally, and allow users to join or leave according to the previous definitions.

Definition 4.6. *A group $\mathbb{G}1_t$ merges with group $\mathbb{G}2_t$, if there is a trace $\tau \in \mathbb{T}$ that both $\mathbb{G}1_t$ and $\mathbb{G}2_t$ can execute such that $\mathbb{G}_{t+i} = \mathbb{G}1_t \cup \mathbb{G}2_t$ (which implies that $\forall\ P \in \mathbb{G}_{t+i}$, $K_{\mathbb{G}_{t+i}} \in \mathbb{P}$), where $K_{\mathbb{G}_{t+i}} = \tau(E^n, M^p)$, $K_{\mathbb{G}_{t+i}} \neq K_{\mathbb{G}1_t}$ and $K_{\mathbb{G}_{t+i}} \neq K_{\mathbb{G}2_t}$.*

Definition 4.7. *A group $\mathbb{G}_t$ splits into groups $\mathbb{G}1_{t+i}$ and $\mathbb{G}2_{t+i}$, if there exists a trace $\tau \in \mathbb{T}$ that $\mathbb{G}_t$ can execute such that $\mathbb{G}_t = \mathbb{G}1_{t+i} \cup \mathbb{G}2_{t+i}$ and $\mathbb{G}1_{t+i} \cap \mathbb{G}2_{t+i} = \phi$ where $K_{\mathbb{G}_t} \neq K_{\mathbb{G}1_{t+n}}$ and $K_{\mathbb{G}_t} \neq K_{\mathbb{G}2_{t+n}}$.*

Some events like split and merge, cannot be executed by normal group members including the dishonest member, but by special members like group leaders. However, a dishonest user can make use of such events when they occur, therefore, we assume that they can be executed by the special member upon the need of the dishonest member. These definitions will be used in order to define rank theorems for the properties we wish to verify.

4.4 Rank Theorems for Protocol Models

Rank functions were first introduced in [11]. For the purpose of establishing the proof that a specific fact will not be available to the intruder, we assign a value or *rank* to each fact, such that, facts that can be generated by the system have positive rank, and facts that cannot be obtained by the intruder cannot have positive rank. The ranks that are assigned will depend on the protocol itself, the initial knowledge and capabilities of the intruder, and the property we want to prove. In our approach, we will define suitable rank functions that map our formal specification requirements in order to obtain rank theorems, which are the properties we wish to prove, the key result that provides the basis of the verification approach is that if these requirements all hold, then no fact of non-positive rank can be generated by the system. Which means that these facts cannot be leaked to dishonest users.

The definition of the rank function is as follows:

Definition 4.8. *(Rank Function) A rank function ρ is a map function $\rho : \mathbb{M} \rightarrow \mathbb{Z}$ which maps the set of all messages into integers.*

The rank function should obey specific rules in order to be sound. First there are no negative ranks generated by the system, $\forall m \in \mathbb{M},\ \rho(m) >= 0$. In order to ensure that facts and signals of positive rank can be generated, it is necessary to verify that each participant cannot introduce anything of non-positive rank to the system. In other words, intruder initial knowledge must be of positive rank, and only facts of positive ranks can be generated from sets of facts of positive rank, $\forall m \in \mathbb{K}_0,\ \rho(m) > 0$. All messages that are supposed to be secret and unknown to the intruder are mapped to zero rank, $\forall m \in \mathbb{S},\ \rho(m) = 0$. When executing an event, the rank of the generated message is a bounded function of the rank of the parameters of the event. For instance, for the *encrypt* event, we define ρ as follows, if $m2 = encrypt(m1, key)$ then $\rho(m2) = \rho(m1) + 1$, where $m1, m2$, and $key \in \mathbb{M}$. Similarly, we define ρ for the *decrypt* event as follows: if $m2 = decrypt(m1, key)$ then $\rho(m2) = \rho(m1) - 1$.

All previous maps define facts about the protocol in general and maps them into integer values. However, for a specific protocol, the rank function may be a little bit different depending on the nature of the protocol. In addition, we can define similar rank functions for all properties we defined in the previous section.

We define a property ϕ for a given group protocol $\mathbb{G}$. This property states that a dishonest user I cannot execute a trace in $\mathbb{T}$ in order to discover a secret in $\mathbb{S}$, and is formally modeled as follows:

$\phi = \forall \tau \in \mathbb{T},\ \tau(E^n, M^p) \rightsquigarrow m \Rightarrow m \notin \mathbb{S}.$

If this property is correct for the protocol $\mathbb{G}$ then we can write $\mathbb{G} \models \phi$.

This is a general secrecy property that will be used to define and proof the rank theorem. The target security property to be verified, i.e., forward secrecy, will be concretely defined later in this section. Now, we define and prove a general rank theorem for this property as follows:

Theorem. *Rank Theorem $\forall m \in \mathbb{K}, \rho(m) > 0 \Rightarrow \mathbb{G}_t \models \phi$, where $m = \tau(E^n, M^p)$ and $\tau \in \mathbb{T}$*

This means that for all traces $\tau \in \mathbb{T}$, a dishonest principal I can execute on a group protocol $\mathbb{G}_t$. We say that the protocol satisfies a security property ϕ,

$\mathbb{G}_t \models \phi$, if the protocol can maintain a positive rank for the messages that can be generated by the intruder.

Proof. We assume there exists $m \in \mathbb{K}$ such that $\rho(m) = 0$, and we show that the property ϕ is invalid.
$\rho(m) = 0 \Rightarrow m \in \mathbb{S}$; $\mathbb{S}$ is a closed set, only messages in $\mathbb{S}$ have rank zero.
$m \in \mathbb{K}$ and $m \notin \mathbb{K}_0 \Rightarrow \exists \tau \in \mathbb{T} : m = \tau(E^n, M^p)$, therefore,
$\tau(E^n, M^p) \rightsquigarrow m$ is valid. Then we can write
$\exists \tau \in \mathbb{T} : \tau(E^n, M^p) \rightsquigarrow m \Rightarrow \rho(m) = 0$, and so
$\exists \tau \in \mathbb{T} : \tau(E^n, M^p) \rightsquigarrow m \Rightarrow m \in \mathbb{S}$, this means
$\rho(m) = 0 \Rightarrow \neg\phi$, and so
$\rho(m) = 0 \Rightarrow \mathbb{G}_t \not\models \phi$

□

Now we define our forward secrecy property, φ, based on the formal specifications model presented previously as follows:
$\varphi = \forall m \in \mathbb{S}, I \in \mathbb{G}_t \ \Rightarrow \neg\exists \tau \in \mathbb{T} : \tau(E^n, M^p) \rightsquigarrow m$

We use the above general theorem in order to define a rank theorem for the forward secrecy property φ (and similarly for backward secrecy). However, we should define the rank function ρ_ϕ that maps the set of all messages to the appropriate ranks. ρ_φ is defined as follows, where $\forall i \in \mathbb{Z} : t + i >= 0$ and $t - i >= 0$.

$$\rho_\varphi(m) = \begin{cases} 0, & if\ m \in \mathbb{S} \ \vee \ m = K_{\mathbb{G}_{t-i}} \\ 1, & if\ m \in \mathbb{K}_0 \ \vee \ m = K_{\mathbb{G}_t} \ \vee \\ & \quad (m = K_{\mathbb{G}_{t+i}} \ \wedge \ I \in \mathbb{G}_{t-i}) \end{cases}$$

This means that for the validity of forward secrecy, we give the rank zero to all the messages in the set of secret messages $\mathbb{S}$, such as secrets shared between users and servers, and all the groups keys that were generated before the current group key. However, for the keys generated after the assumed dishonest used joined the group are mapped to a positive rank because they are in his initial set of knowledge.

Now we can write the above theorem for forward secrecy property as follows

Theorem. *Rank Theorem for Forward Secrecy Property:* $\forall m \in \mathbb{K},\ \rho_\varphi(m) > 0 \Rightarrow \mathbb{G}_t \models \varphi$*, where* $m = \tau(E^n, M^p)$ *and* $\tau \in \mathbb{T}$

Similarly, we define a rank function and then a theorem for backward secrecy property as follows:

$$\rho_\psi(m) = \begin{cases} 0, & if\ m \in \mathbb{S} \ \vee \ m = K_{\mathbb{G}_{t+i}} \\ 1, & if\ m \in \mathbb{K}_0 \ \vee \ m = K_{\mathbb{G}_t} \ \vee \\ & \quad (m = K_{\mathbb{G}_{t-i}} \ \wedge \ I \in \mathbb{G}_{t-i}) \end{cases}$$

Similarly, for the validity of backward secrecy, we give the rank zero to all the secret messages and all the groups keys that are generated after the current group key. For these keys generated before the assumed dishonest used leave the group are mapped to a positive rank because they are in his initial set of knowledge.

Theorem. *Rank Theorem for Backward Secrecy Property:* $\forall m \in \mathbb{K},\ \rho_{\psi}(m) > 0 \Rightarrow \mathbb{G}_t \models \varphi$, *where* $m = \tau(E^n, M^p)$ *and* $\tau \in \mathbb{T}$

One of the advantages of introducing such theorems, is that, first, it is protocol independent, which means that we can apply it on different protocols as well as on the same protocols at different levels of abstraction. Second, it is implementation independent, which gives more freedom to verification tool choice without any modification on the previous steps of our methodology.

5 PVS Implementation

The last step of our methodology is to mechanize the proof of the rank theorem using a verification tool, for this purpose we choose the PVS theorem prover. Our model in the PVS includes an embedding of the formal requirements that we defined for the events and traces of execution, the rank functions and the rank theorem of the security property. Then we prove in PVS that the rank theorem maintains a positive rank, which implies the correctness of the property with respect to the protocol. In addition, we need to consider timing since security properties we deal with are time dependent (such as forward and backward secrecy), and protocols messages may contain time-stamps as part of the message.

In our implementation, we first formalized the general requirement, rank function and its lemmas, and the rank theorem. First we show the type declarations we used for our model, this includes the types of messages, events, key, a subtype of messages, and users, traces, and groups. Actually, the type message is defined as a record that contains all the components of messages such as source of message, intended destination of message, key used for encryption, and nonces.

```
MESSAGE : TYPE
EVENT : TYPE = MESSAGE, MESSAGE -> MESSAGE
KEY : TYPE FROM MESSAGE
USER : TYPE
TRACE : TYPE = set[EVENT]
GROUP : TYPE
```

Then we define the sets of messages we use in our model, including the set of all messages, secret messages, events, traces, intruders initial knowledge, intruders updated knowledge.

```
    allmsgs: VAR set[MESSAGE]
    allEvents: VAR set[EVENT]
    secretKey: VAR set[KEY]
    traces: VAR set[TRACE]
    allEvents: VAR set[EVENT]
    intInitKnldg : set[MESSAGE]
    intKnldg : set[MESSAGE]
```

We define a number of variables for the users of the protocol, including normal users, and leaders, in addition to the intruder. In our model, we abstract the network since it has no effect based on the assumptions made about the intruder.

```
    I : VAR USER
    U : array[n_users] of USER
    L : array[n_leaders] of LEADER
```

Then we defined the prototypes for the events protocols can execute, this includes the normal events, like send, receive, encrypt, and decrypts, in addition to the dynamic events such as join, leave, merge split. These events of the protocol are represented in PVS as a data type in order to be sure that all actions are syntactically different.

```
Event : DATATYPE
   BEGIN
      send(msg_send, m_recv: USER,
              s_msg: MESSAGE): send?
      recv(m_recv, m_send: USER,
              r_msg: MESSAGE) : recv?
      join(user: USER, group : GROUP) : join?
      leave(user: USER, group : GROUP) :leave?
      merge(x_group, y_group : GROUP): merge?
      split(group: GROUP) : split?
   END event
```

In order to define the rank function for forward secrecy property we use the predicate $inSet$ which tells if a given message belongs to a specific set of messages. This predicate is defined as follows:

Now we can define the rank function for forward secrecy property that initializes every message in the intruders initial set of knowledge and all the messages in the set of secret messages, this definition represents the initialization of the ranks of the messages in the initial state of the protocol, when executing the protocol,

```
inSet: [set[MESSAGE], MESSAGE -> bool] =
  (LAMBDA (p: set[MESSAGE], m: MESSAGE): p(m)
```

every new generated message will have a specific rank thats calculated depending on the events executed.

```
rank(msg:MESSAGE) : NAT =
   IF msg = id THEN 1
   ELSEIF msg = nonce THEN 1
   ELSEIF inSet(secretKey,m) THEN 0
   ELSEIF inSet(intInitKnldg,m) = THEN 1
   ENDIF
```

Next, we show how we update ranks of newly generated messages from events in PVS as follows:

```
updateRank(event,m1,m2, key, u1,u2) : nat =
   CASES event OF
      concat(m1,m2)  : MIN(rank(m1),rank(m2)
      encr(m1,key)   : rank(m1)+1
      decr(m1,key)   : rank(m1)-1
      send(u1,u2,m1) : rank(m1)
      recv(u1,u2,m1) : rank(m1)

   ENDCASES
```

At this stage, we need lemmas for the rank functions consistency. Since rank functions should meet specific requirements in order to be consistent and guarantee the correctness of the rank theorem, we state lemmas that ensure the correctness of the rank function. The first lemma states that there are no negative ranks generated by the system for any message.

```
BEGIN
  m1: VAR MESSAGE
  non_neg_rank: LEMMA
    FORALL m1: rank(m1) >= 0
END
```

The second lemma states that when executing an event, the rank of the generated message is bounded by the rank of the message(s) used by the event, in other words, the rank of the new message maintains the same value of the rank of the previous message, incremented by one, or decremented by one.

```
BEGIN
  m1: VAR MESSAGE
  m2: VAR MESSAGE
  e1: VAR EVENT

  bounded_rank: LEMMA
    m1 = e1(m2) IMPLIES
    rho(m1) = rank(m2) OR
    rho(m1) = rank(m2) + 1 OR
    rho(m1) = rank(m2) - 1
END
```

The last lemma states that if applying two evens in sequence will result in the original message (i.e., inverse events), then the rank of the message after applying these two events should remain the same. These lemmas are necessary to guarantee that when a zero rank is generated, it is actually generated by executing a trace of events in the protocol, not by an inappropriate map or inconsistency in the rank function definition.

```
BEGIN
  m1: VAR MESSAGE
  m2: VAR MESSAGE
  e1: VAR EVENT
  e2: VAR EVENT

  inverse_event: LEMMA
     m1 = e1(m2) AND m3 = e2(m2) AND m1 = m3
        IMPLIES rank(m1) = rank(m3)
END
```

At this point we encoded our forward secrecy property and our rank theorem for this property in PVS as follows:

This is the basic lemma we proof based on previous definition. The set *intKnldg* is updated by the intruder, and for ever update we calculate the new rank as shown above. So the proof means that the intruder who executes any of the above defined events for the protocol cannot obtain a message with rank zero.

```
forward_secrecy : THEORY
  BEGIN

      msg  : VAR MESSAGE
      fwd_secrecy_property: LEMMA
         FORALL msg: inSet(intKnldg,msg)
            IMPLIES rank(msg) > 0

  END forward_secrecy
```

6 Application: Enclaves Protocol

Enclaves [3] is a protocol that enables users to share information and collaborate securely through insecure networks such as the Internet and provides services for building and managing groups of users. Authorized users can dynamically, and at their will, join, leave, and rejoin an active group. The group communication service relies on a secure multicasting channel that ensures integrity and confidentiality of group communication. The group-management service consists of user authentication, access control, and group-key distribution. We apply our approach on this protocol and show the correctness of its forward secrecy property.

A user who is willing to join the group sends requests to a set of leaders. The leaders locally authenticate the user, and establish an agreement protocol among them, as whether or not to accept the user. Upon acceptance, the user is provided with the current group composition, as well as information to construct the group-key. Each member is notified when a new user joins or a member leaves the group in such a way that all members are in possession of a consistent image of the current group-key holders.

In our PVS model of the protocol we made some assumptions about the protocol, including abstracting the hash functions and the mathematical computations in secret key calculation. We also assumed that the group member can compute the secret key only if he/she has all the necessary secret shares from group leaders, this fact is imposed by the group key management of the protocol.

Concerning the dishonest user, we assumed, they can monitor the network and chooses to send messages on the network, either randomly or at their choice. However, they cannot block messages from reaching their destination. Finally, we assumed they are limited by cryptographic constraints. For instance, they cannot decrypt messages without having the key, or impersonate other participants by forging cryptographic signatures.

The implementation was applied on the forward secrecy property, following the same steps, backward secrecy property can verified following the same step.

We formalized the protocol events in PVS, utilizing previous implementations by [5] and [3], including all the operations that can be executed on the group. In following, we show parts of the PVS implementation, which consists of a number of steps executed by users and leaders, a number of reachable states, and a number of PVS propositions to reason about certain activities, like group key possession and joined status.

The first part of the code shows the concrete variables declarations that are used in the implementation of the PVS protocol model.

```
BEGIN
  A, B, C: VAR USER
  G: VAR GROUP
  S: VAR set[MESSAGE]
  T: VAR TRACE
  N, Nl, Nl1, Nl2, Na, Na1, Na2 : VAR NONCE
  K, Ka, Kg, Kb: VAR KEY
  q, q1, q2, q3: VAR global
  e, e1, e2: VAR EVENT
  n, n1, n2: VAR nat
```

Then we show partially the execution steps of the protocol taken by a user and leaders in order to complete the protocol.

```
  step_01(q): bool =
    q‘users(A0) = NotConnected and
    q‘leader(A0) = NotConnected

  step_02(q): bool =
    EXISTS Na:
      q‘users(A0) = WaitingForKey(Na) &
      q‘leader(A0) = NotConnected &
      (FORALL K, N: not PartsTrace(q)
      (Encr(Shr(A0),Leader ++A0 ++Na ++N ++K)))

  step_03(q): bool =
    EXISTS Na, Nl, Ka:
      q‘users(A0) = WaitingForKey(Na) &
      q‘leader(A0) = WaitingForKeyAck(Nl, Ka) &
      (FORALL K, N: PartsTrace(q)(Encr(Shr(A0),
      Leader ++A0 ++Na ++N ++K))=> N=Nl & K=Ka)
      & (FORALL N: not PartsTrace(q)
        (Encr(Ka, A0 ++ Leader ++ Nl ++ N))) &
      not PartsTrace(q)(Encr(Ka, A0 ++ Leader))
```

Next, we define, as lemmas, the reachable states in the protocol, where, for every one, there is a set of conditions to be satisfied.

```
tran_01: LEMMA
  Reachable(step_00, T)
     (q1) AND step_01(q1) AND T(G)(q1, e, q2)
        IMPLIES step_01(q2) OR step_02(q2)
             OR step_12(q2)
tran_02: LEMMA
  Reachable(step_00, T)
    (q1) AND step_02(q1) AND T(G)(q1, e, q2)
       IMPLIES step_02(q2) OR step_03(q2)
            OR step_013(q2)
tran_03: LEMMA
  Reachable(step_00, T)
    (q1) AND step_03(q1) AND T(G)(q1, e, q2)
       IMPLIES step_03(q2) OR step_04(q2)
```

Then we define a preposition to show the possession of a key by a specific user after executing the necessary steps. Then we describe the connection state of a user which indicates that a user is connected to the group if all the given premises are valid.

```
session_key_prop: PROPOSITION
  Reachable(step_00, T)(q) AND q‘users(A0) =
    Joined(N, Ka) => InUse(Ka, q)

joined_states: PROPOSITION
  Reachable(step_00, T)(q) AND
  Joined?(q‘users(A0)) AND
  Joined?(q‘leader(A0))
  => EXISTS Ka, Na: q‘users(A0) =
    Joined(Na, Ka) AND q‘leader(A0) =
    Joined(Na, Ka)
END
```

At this point, we can instantiate our rank theorem for forward secrecy and check its validity in the protocol states. Which means that starting from the first step in the protocol, the initial step, and applying any trace, the rank of any message in the intruder knowledge is positive.

The rich datatype package of PVS helped in formalizing the protocol requirement in a way that thoroughly captures the many subtleties on which the correctness arguments of the protocol rely. The PVS theorem prover provides a collection of powerful primitive inference procedures to help derive theorems, where higher level proof strategies can be defined in order to make the verification process easier. This will allow similar theorems to be proved efficiently, therefore; we can apply our methodology on similar properties like backward secrecy property effi-

```
forward_secrecy : THEORY
  BEGIN
      fwd_secrecy: LEMMA
         FORALL m,T: Reachable(step_00, T)
                 AND inSet(intKnldg,m)
                     IMPLIES rank(m) > 0
  END forward_secrecy
```

ciently and directly.

Using the features of PVS, we have proved that the protocol satisfies forward secrecy property by establishing the correctness of the above theorem *Rank Theorem for Forward Secrecy Property* in PVS. The proof was conducted using the set of general requirements in addition to the protocol model, the implementation of the proof took around three months. The proof of backward secrecy can be derived in a similar fashion, and in much shorter time, given the experience gained.

7 Conclusion

The correctness of security protocols in communication systems remains a great challenge because of the sensitivity of the services provided. Formal methods have been used widely in this area to perform protocol verification and analysis. In this paper, we illustrated the need for a verification methodology for a class of protocols that deal with group key distribution. While most approaches in the literature target cryptographic properties for two parties protocols, the verification problem for group key distribution protocols is more challenging. In addition, properties like forward and backward secrecy are very important for protocols correctness, however, they did not receive enough attention in the literature.

The contributions of this paper are providing a set of generic requirements of group key distribution protocols, then establishing their formal specifications, then define a formal model for this class of protocols, and finally present rank theorems to enable and mechanize the verification procedure of this class of protocols. We used rank functions that map the formal protocol model into a set of integers in order to obtain rank theorems, then embedded our model, rank functions and rank theorems in PVS in order to construct the proof of the claimed security properties. We proof the soundness of our approach by proving the correctness of the rank theorem. We applied our methodology on the Enclaves group management protocol, and constructed the correctness proof for forward secrecy property for this protocol in PVS. We are in the process of extending the above approach to prove other properties for the same protocol, like backward secrecy property, authentication, and group consistency under protocol events.

As immediate future work, we plan to elaborate on the requirements for the rank function choice, and then establish the correctness of rank functions by showing their coherence and consistency. Once we prove that a rank function with specific requirements is correct, we can set the rules to choose appropriate rank functions for the security property of interest. This will provide us with a formal link between the rank theorems and the security property under investigation.

We also plan to extend this approach by using an inference system along with the rank functions. The inference system represents the events of the protocols. Another open issue is applying abstraction techniques on the rank function to be able to model them in first order logic, and therefore, make model checking approach feasible.

References

[1] M. Archer. Proving Correctness of the Basic TESLA Multicast Stream Authentication Protocol with TAME. In *Workshop on Issues in the Theory of Security*, January 2002.

[2] G. Denker and J. Millen. Modeling Group Communication Protocols using Multiset Term Rewriting. In *Rewriting Logic and its Applications*, volume 71 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[3] B. Dutertre, V. Crettaz, and V. Stavridou. Intrusion-Tolerant Enclaves. In *Proc. IEEE International Symposium on Security and Privacy*, pages 216–224, May 2002.

[4] B. Dutertre and S. Schneider. Using a PVS Embedding of CSP to Verify Authentication Protocols. In *Theorem Proving in Higher Order Logics*, volume 1275 of *Lecture Notes in Computer Science*, pages 121–136. Springer-Verlag, 1997.

[5] M. Layouni, J. Hooman, and S. Tahar. On the Correctness of an Intrusion-Tolerant Group Communication Protocol. In *Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 231–246. Springer-Verlag, 2003.

[6] C. Meadows and P. Syverson. Formalizing GDOI Group Key Management Requirements in NPATRL. In *Proc. ACM Conference on Computer and Communications Security*, pages 235–244, November 2001.

[7] C. Meadows, P. Syverson, and I. Cervesato. Formal Specification and Analysis of the Group Domain of Interpretation Protocol using NPATRL and the NRL Protocol Analyzer. *Journal of Computer Security*, 12(6):893–932, 2004.

[8] S. Owre, J.M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer Verlag, 1992.

[9] O. Pereira and J. Quisquater. Some Attacks upon Authenticated Group Key Agreement Protocols. *Journal of Computer Security*, 11(4):555–580, 2004.

[10] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient Authentication and Signing of Multicast Streams over Lossy Channels. In *Proc. IEEE Symposium on Security and Prvacy*, pages 56–73, Washington, DC, USA, May 2000.

[11] P. Ryan and S. Schneider. *The Modelling and Analysis of Security Protocols: The CSP Approach*. Addison-Wesley, 2001.

[12] G. Steel, A. Bundy, and M. Maidl. Attacking a Protocol for Group Key Agreement by Refuting Incorrect Inductive Conjectures. In *Automated Reasoning*, volume 3097 of *Lecture Notes in Computer Science*, pages 137–151. Springer-Verlag, 2004.

[13] H. Sun and D. Lin. Dynamic Security Analysis of Group Key Agreement Protocol. *IEEE Transactions on Communication*, 152(2):134 – 137, April 2005.

[14] P. Syverson and C. Meadows. Formal Requirements for Key Distribution Protocols. In *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 320–331. Springer-Verlag, 1995.

[15] T. Truderung. Selecting Theories and Recursive Protocols. In *Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 217–232. Springer-Verlag, 2005.