On the Formal Verification of Group Key Security Protocols

Amjad Gawanmeh

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy at

Concordia University

Montréal, Québec, Canada

September 2008

CONCORDIA UNIVERSITY

Division of Graduate Studies

This is to certify that the thesis prepared

By: **Amjad Gawanmeh**

Entitled: **On the Formal Verification of Group Key Security Protocols**

and submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

________________________________ Dr.

________________________________ Dr. Eltoweissy, Mohamed

________________________________ Dr. Haghighat, Fariborz

________________________________ Dr. Youssef, Amr

________________________________ Dr. Ait Mohamed, Otmane

________________________________ Dr. Tahar, Sofiène

Approved by ____

Chair of the ECE Department

__________ 2008 ____

Dean of Engineering

ABSTRACT

On the Formal Verification of Group Key Security Protocols

Amjad Gawanmeh, Ph.D.

Concordia University, 2008

The correctness of group key security protocols in communication systems remains a great challenge because of dynamic characteristics of group key construction as we deal with an open number of group members. Therefore, verification approaches for two parties protocols cannot be applied on group key protocols. 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 secrecy, which has more complex variations in group key context: forward secrecy, backward secrecy, and key independence.

In this thesis, we present a combination of three different theorem-proving methods to verify security properties for group-oriented protocols.We target regular group secrecy, forward secrecy, backward secrecy, and collusion properties for group key protocols. In the first method, rank theorems for forward properties are established based on a set of generic formal specification requirements for group key management and distribution protocols. 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. Rank theorems can only reason about absence of attacks in group key protocols. In the second method, a sound and complete inference system is provided to detect attacks in group key management protocols. The inference system provides an elegant and natural proof strategy for such protocols compared to existing approaches. It complements rank theorems by providing a method to reason about the existence of attacks in group key protocols. However, these two methods are based on interactive higher-order logic theorem proving, and therefore require expensive user interactions. Therefore, in the third method, an automation sense is added to the above techniques by using an event-B first-order theorem proving system to provide invariant checking for group key secrecy property and forward secrecy property. This is not a straightforward task, and should be based on a correct semantical link between group key protocols and event-B models. However, in this method, the number of protocol participants that can be considered is limited, it is also applicable on a single protocol event. Finally, it cannot model backward secrecy and key independence. We applied each of the developed methods on a different group protocol from the literature illustrating the features of each approach.

To My Parents.

ACKNOWLEDGEMENTS

First, I would like to express my gratitude to my supervisor, Dr. Sofiène Tahar, for his strong support, encouragement and guidance through out my Ph.D study. He was always approachable and his insights about research and immense knowledge in the field of formal methods have strengthened this work significantly. I would also like to acknowledge his efforts for establishing a very friendly and stimulating atmosphere in the Hardware Verification Group (HVG).

I would like to thank Dr. Adel Bouhoula from Higher School of Communication of Tunis in Tunisia and Dr. Leila Jemni Ben Ayed from Research Laboratory of Technologies of Information and Communication in Tunisia for the technical collaboration, discussions, and feedback on the research project during their visits to HVG.

I would like to thank all members of my committee, Dr. Mohamed Eltoweissy from Virginia Tech university for serving as my external examiner, Dr Fariborz Haghighat, Dr. Amr Youssef and Dr. Otmane Ait Mohamed from Concordia University. Also I would like to thank Dr. Ferhat Khendek, Dr. Abdeslam En-Nouaary, and Dr. Radhika Katarzyna for their past serving in my PhD advisory committee.

Very special thanks go out to my colleagues in the Hardware Verification Group (HVG), current and previous members. Without their help, motivation and encouragement, I would not have reached this point. I would like to thank all my friends here in Montreal and everywhere else, their encouragement and support was something I will always appreciate.

Finally, I would like to thank my sisters and brothers for the support they always provide me with. Most importantly, I wish to thank enough my parents, Mofeedeh and Mahmoud: they raised me, supported me, taught me, and loved me. To them I dedicate this thesis.

Thanks to Allah for everything.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

A-GDH	Authenticated Group Diffie-Hellman protocol
AKA	Auxiliary Key Agreement
AKE	Authenticated Key Exchange
CSP	Communication Sequential Processes
DH	Diffie-Hellman
DoS	Denial of Service
FDR	Failure Divergences Refinement
GDH	Group Diffie-Hellman
GDOI	Group Domain of Interpretation
GKMP	Group Key Management
HOL	Higher Order Logic
IKA	Initial Key Agreement
IKE	Internet Key Exchange
IKEv1	Internet Key Exchange version 1
NSPKP	Needham-Schroeder Public-Key
NSSK	Needham-Schroeder Secret-Key
NPATRL	NRL Protocol Analyzer Temporal Language
NRL	Naval Research Laboratory (Protocol Analyzer)
OFT	One-way Function Tree
PKI	Public Key Infrastructure
PVS	Prototype Verification System
SET	Secure Electronic Transactions
TAME	Timed Automata Modeling Environment

TCCs	Type-Correctness Conditions
TGDH	Tree-based Group Diffie-Hellman
UML	Unified Modeling Language

Chapter 1

Introduction

1.1 Motivation

A typical security system consists of a number of principals such as people and computers, that communicate using a variety of channels. The security protocols are the rules that govern these communications. They are typically designed so that the system will survive malicious acts. Protocols are designed under certain assumptions about the threats because protection against all possible attacks is too expensive. Protocols may be extremely simple or very complex. For instance, the networks of cash machines have dozens of protocols specifying how a cash machine interacts with customers, how it talks to the bank that operates it, how the bank communicates with the network operator, how money gets settled between banks, how encryption keys are set up between the various principals, and what sort of alarm messages may be transmitted, such as instructions to capture a card. All these protocols have to work together in a large and complex system, and most important, they have to guarantee the security of communications when dealing with critical issues [5].

The meaning of “protocol” is: a prescribed sequence of interactions between entities designed to achieve a certain goal. A protocol with security objective is called security protocol. Security protocols in particular shall provide security properties of distributed systems. Cryptographic protocols are security protocols that use cryptographic mechanisms such as encryption algorithms and digital signature schemes as basic components.

Group communication applications use encryption methods in order to limit access to information for legitimate members only. They are also dynamic with regard to principals participating in the group. So messages are protected by encryption using a chosen key, which in the context of group communication is called the group key. Only those who know the group key are able to recover the original message. In addition, the group may require that membership changes cause the group key to be refreshed. Changing the group key prevents a new member from decoding messages exchanged before they joined the group or after they leave. If a new key is distributed to the group when a new member joins, the new member cannot decipher previous messages even if it has recorded earlier messages encrypted with the old key. Distributing the group key to legitimate members is a complex problem. A group key distributor must provide another scalable mechanism to distribute keys to the legitimate principals and must guarantee the secrecy of these group keys.

The general requirements for protocols involving two or three parties [20] are well understood, however, the case is different with group key 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 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 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. This fact results in the concepts of forward secrecy, backward secrecy and key independence (or collusion resistance) properties. 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.

There are many examples in the literature of protocols that have been used extensively before it turns out that an attack can be taken against the protocol, even though the protocol received intensive analysis, and thought to be correct before they were found to be flawn. For instance, the Needham-Shroeder authenticated key distribution protocol [65], which was found to allow an intruder to pass an old, compromised session key as a new one to a legitimate party [54]; a protocol in an early draft of the CCITT X.509 draft standard [21], for which Burrows, Abadi, and Needham [18] showed that an intruder can cause an old session key to be accepted as a new one, whether or not it had been compromised; Pereira and Quisquater [69] pointed out two attacks on the Group Diffie-Hellman protocol [7]; another example is an attack on the Internet Key Exchange (IKE) protocol was found independently by Ferguson and Schneier [38] and Zhou [87]. Kats and Shin [45] addressed the case of attacks by malicious insiders for authenticated key exchange protocols. Pereira and Quisquater [70] provided a systematic way to derive an attack against any Authenticated Group Diffie-Hellman (A-GDH) type protocol with at least four participants and exhibit protocols with two and three participants.

We also have noticed that there is an increasing interest in deploying group key protocols in wireless networks such as the work in [64] and wireless sensor networks such as the work in [63] and [22]. This brings more motivation for the need for correct and secure group key protocols, and therefore, new efficient and scalable verification techniques should be adapted.

These examples show that the informal design of cryptographic protocols is error prone because reasoning about them is extremely difficult. This motivated people to use formal methods [24] not only in the analysis and verification of protocols, but also in the design of these protocols. So, cryptographic protocols are excellent candidates for rigorous formal analysis. They are critical components of distributed security, very easy to express and very difficult to evaluate by hand. Formal techniques can be used in various phases of the design of a cryptographic protocols including the specification, the construction, and the verification.

Most of the existing verification methods in the literature deal with secrecy and authentication properties for two parties protocols. However, the time needed for verifying of a protocol using tools is still exponential with respect to the number of messages. Consequently, automatic verification of large protocols is infeasible without simplifications, assumptions and abstractions. The verification of a multi-party security protocol by a model checking [25] approach is even more problematic. It can only be done for instances with a fixed number of users and fixed number of messages. In addition, it is infeasible to model certain features of these protocols such as events, unbounded messages space, or unbounded group members, and reason about using model checking approaches. Therefore, in order to analyze a group key protocol in its full generality, theorem proving techniques [40] are required.

Theorem proving is a formal method where the specification and implementation are expressed in first-order or higher-order logic. Then, relationship between the two models is formed as a theorem to be proven within a deduction logic system. Formal methods have not been used efficiently for verifying group key protocols, specially for properties like forward and backward secrecy. In this thesis, we address the verification of group key protocols, targeting properties that are specific for this class of protocols.

1.2 Group Key Security Protocols

Group key management protocols are used to generate and distribute keys. In this thesis we use the term “group key protocols” or “group key security protocols” to refer to group key management protocols.

The goals of security protocols are to provide security services for communicating entities. The protocol involves a precise interaction between the entities in order to achieve the required service, and sometimes a trusted third party is involved in this interaction. There are two classes of protocols: authentication protocols which allow users to identify each other, and key management protocols which aim to generate and distribute cryptographic keys between principles.

The security services provided by security protocols were presented in [75], and can be intuitively summarized as:

1. *Data Integrity*: the receiver of the message should be able to find out whether the message was modified during transmission. Modification can occur accidentally, or intentionally. No one should be able to substitute a false message for the original message or part of it.

2. *Authentication of origin*: which means that we can be sure that a message we believe it was originated from a certain node was indeed originated from that node.

3. *Establishing session keys between nodes*: Keys are considered sensitive data items that should be revealed only to legitimate users.

4. *Secrecy (or confidentiality)*: which means that the intruder should not be able to deduce anything about the legitimate users' activity. Provided that an intruder is able to read the message, he should not be able to derive its contents.

5. *Non-repudiation*: which is providing parties with evidence that certain steps of the protocol has occurred to protect them from cheating by other users we assume to be honest.

Other services that can be provided by security protocols include fairness, anonymity, availability, protection against denial of service, and resistance to traffic analysis. *Fairness*, means that none of the participants in the protocol is able to gain some advantage over another, for example by refusing to continue after the other end has signed up, before he has signed. *Anonymity* means that an observer will not be able to identify events that occur during protocol runs, even though he/she can be able to deduce its occurrence. *Availability* means that the protocol should achieve some desired goal. In *denial of service attacks*, an tacker initiates an instance of a protocol and then drops out leaving the victim hanging, by initiating and then dropping enough instances of the protocol, the victim will run out of resources [58]. *Traffic analysis* is performed by intercepting and examining messages and their routing in order to deduce information from patterns in communication [36].

A protocol can be designed to provide one or more of these services. The problem of formally modeling these services received intensive discussion in [75]. More important is how to proof that the protocol satisfy security measures imposed by these services.

As networking becomes more widespread and handle more and more tasks in a potentially hostile environment, protocols should deal with this environment. Therefore, new protocols emerged like the Internet Key Exchange (IKE) protocol [41] and its latest version IKEv1 [43], that not only must agree upon encryption keys, but on the algorithms to use. Another example is the Secure Electronic Transaction (SET) protocol [56] that must be able to process different types of credit card transactions. This of course caused increasing the complexity of protocols and made verification and implementation of the protocol more difficult.

The encryption, hashing, modulo and exponentiation methods are used in security protocols and are known as *algebraic* properties of the protocol. In order to prove the correctness of group key protocols, rigorous analysis of these protocols are mostly used focusing on these algebraic properties. However, protocols can be subtle to *non-algebraic* attacks where the intruder takes advantage of the distributive nature of these protocols. Therefore, it is essential to prove the security of these protocol from both algebraic and non-algebraic points of view.

1.3 Verification of Group Key Protocols: State-of-the-Art

In this section we review and discuss approaches for modeling and verification of group key protocols, focusing more on those that are closely related to our work. The state of the art methods are classified into two major categories: the first category deals with analysis methods to verify group key protocols, and the second category deals with formal methods for group key protocols.

The formal approach to security protocols uses simple logical reasoning. The analysis approach, on the other hand, makes a delicate use of probability and computational complexity. These two approaches have distinct views of primitive cryptographic and computational operations at different levels of abstraction. In the formal approach, cryptographic operations are formally modeled as functions on a space of symbolic expressions; their security properties are also modeled formally. This is done at a high level of abstraction. In the analysis approach, cryptographic and computational operations are seen as functions on strings of bits or algebraic expressions; their security properties are defined in terms of the probability and computational complexity of successful attacks [1].

1.3.1 Analysis Methods

Pereira and Quisquater [69] proposed a systematic approach to analyze protocol suites extending the Diffie-Hellman key-exchange scheme to a group setting. They 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 GDH protocols. In a more recent work Pereira and Quisquater [70] provided a systematic way to derive an attack against any Authenticated GDH (A-GDH) type protocol with at least four participants and exhibit protocols with two and three participants.

Truderung [83] presented 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, however, cannot be applied on non–recursive protocols such as GDH or the Enclaves. In addition, the model provided is not readable and very complex to construct.

Bresson *et al.* [14] presented a security model Group Diffie-Hellman protocols for Authenticated Key Exchange (AKE) and used it to precisely define AKE and the entity-authentication goal as well. Then, they defined in this model the execution of an authenticated group Diffie-Hellman scheme and proved its security. In more recent efforts, Bresson *et al.* [15, 16] discussed the GDH problem thoroughly and suggested a model for this class of protocols in the presence of malicious participants.

Horn [44] showed that the One-way Function Tree (OFT) protocol proposed by Sherman and McGrew [77] is subject to a particular kind of collusion attack that was pointed out in which was solved in an improved version by Ku and Chin [51]. Recently, Xu *et al.* [86] generalized the example attack to a generic collusion attack on OFT class of protocols. They provided necessary and sufficient conditions for this attack to exist. Then they suggested a solution to prevent the attack with enhanced performance over previous solutions.

These methods are appropriate for the analysis of algebraic related properties of protocols. However, there are examples of protocols that have been used extensively before it turns out that an attack can be taken against them, even though the protocol received intensive analysis, but the flaw could not be addressed. This triggered people towards using formal methods to design and verify protocols. There are many approaches and tools that have been developed in this direction, these are discussed next.

1.3.2 Formal Verification Methods

Formal methods use a combination of a mathematical or logical model of a system and its requirements, together with an effective procedure for determining whether a proof that a system satisfies its requirements is correct. Formal methods have gained attention in the analysis of security protocols since the early trials of proving that protocols are secure. Some flaws in famous protocols have been found using formal methods, where these protocols received intensive analysis before. This motivated people to use formal methods not only in the analysis and verification of protocols, but also in the design of these protocols.

The last years have seen the emergence of successful applications of formal approaches to reasoning about security protocols. Earlier methods were concerned with reasoning about the events that a security protocol can perform, and make use of a causal dependency that exists between protocol events. Methods like strand spaces [37] and the inductive method of Paulson [68] have been designed to support an intensional, event-based, style of reasoning. These methods have successfully tackled a number of protocols. Syverson and Meadows [82] presented the formal requirements for authentication in key distribution protocols to provide a single set of requirements to specify a whole class of protocols, which can be fine-tuned for the particular application. Syverson and Meadows extended this work in [59] and 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 [8]. In a subsequent work, Meadows *et al.* [60] 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 no general set of requirements for protocols requirements can be applied on a specific protocol, or can be used for the refinement of protocol specifications during the design process. In addition, the requirements they provide were 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.

Dutertre and Schneider [35] first introduced and used *rank functions* for the embedding of CSP (Communication Sequential Process) in PVS in order to verify the authentication property of Needham-Shroeder public key protocol. They proposed the idea of rank functions in order to enable CSP verification of the Needham-Shroeder protocol. Later, Schneider [75] used the idea of rank functions for the verification of CSP. The work did not present a method that can be applied on security properties in other classes of protocols, specifically, group key protocols. In fact, the method, as is, may not be applied on secrecy property for group key management protocols. In [30] Delicata and Schneider described how their rank function approach is incapable of proving the correctness of protocols that are modeled using the process algebra CSP against forward secrecy property. This shortcoming motivated them to propose another proof technique based on the concept of a temporal rank function, where they use time stamps to tag messages and provide rank for these tags. Even though the method is novel in its treatment for forward secrecy property, the method considers partial forward secrecy since it provides only three ranks for temporal messages: $0, n,$ or ∞. Delicata and Schneider [31] present an algebraic approach for reasoning about secrecy in a class of Diffie-Hellman protocols. The technique uses the notion of a message template to determine whether a given value can be generated by an intruder in a protocol model. The work is restricted to certain algebraic form of messages that are expressible as g raised to the power of a sum of products of integers, and therefore requires further extension to handle messages with different algebraic structures. The authors relaxed the restrictions on the algebra of the protocol to allow the expression of its messages. This is an interesting method, however it is based on establishing a hierarchy of secrecy, where a message that can be leaked in the future is less secret than another message. However, this concept may not be applied on backward secrecy and henceforth collusion properties.

Layouni *et al.* [53] used a combination of three different approaches to prove the correctness of Intrusion-tolerant Enclaves protocol [34, 33]. They used the Murphi model checker in order to verify authentication property, and PVS to verify safety and liveness properties such as proper agreement, agreement termination, and integrity, finally, they used a Random Oracle model to manually proof robustness and unpredictability properties.

The choice of the techniques was driven by the nature of the correctness arguments in each module of the protocol, by the environment assumptions and the capability of the verification approaches. This example shows that it is difficult to verify and analyze this class of protocols. There was no single formalism where the protocol can fit and its verification is feasible. However, the authors achieved a promising success in verifying a complex protocol such as Enclaves, the results could be improved further first by using the rank functions to partition the message space and mechanically proof properties for the protocol [75], second by providing correctness proofs for the consistency group membership when members leave and join the group. Finally one of the biggest challenges is to perform the analysis of the group key management module in PVS which requires the elaboration of some general purpose theories that deal with probabilities which are not yet available in theorem provers. 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.

In another work, Sun and Lin [81] extended the strand space theory [37] 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. While this work provides a method to verify complex group protocols, the proposed solutions focus only on specific aspects of one protocol rather than focusing on general requirements.

Cremers [28] proposed an operational semantics for security protocols. The work provides a generic description of the interpretation of such security protocols and what it means for a protocol to ensure some security property. This work imposes explicit static requirements for valid protocols, and verifies that the model is parametric with respect to the matching function and intruder network capabilities.

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. Most approaches focus on 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 protocol. We propose a framework to fill this gap by presenting new methods to verify group key protocol specific properties, as it will be presented in the next section.

Some other recent works provide analysis and computational models for GDH like protocols neglecting machine assisted verification techniques, such as theorem proving. In addition, the focus on rigorous analysis of security of these protocols neglects possible source of protocols vulnerabilities, where the intruder can take advantage of the distributive nature of these protocols. Therefore, it is essential to prove the security of these protocol from both algebraic and non-algebraic points of view.

1.4 Proposed Verification Methodology

We propose a framework for the formal specification and verification of group key security protocols. In this framework, we first define the specification requirements for group key protocols. Based on these requirements, we need to define a formal model with well-defined semantics that can handle group key protocols and specify their security properties. This model should capture all specification requirements for group protocols such as the handling of large, or even unlimited, number of participants, capturing the increasing different kinds of environments within which the protocol works, capturing new protocols requirements and properties: like forward and backward secrecy and immunity to collusion attacks, and finally, providing methods for validation and verification in order to be able to reason about the correctness of the protocol, and to prove its security against specific attacks. In addition, these methods should be able to detect attacks if they exist.

The next step is to define verification methods that can handle the class of protocols under investigation, and verify the required security properties for group key protocols. The proposed verification methods should be able to prove the correctness of the group protocol under investigation, verify security properties provided by the protocol, mainly safe key distribution, and finally, handle a protocol model with abstracted features. In addition, these methods should be efficiently implemented, i.e., their complexity should be reasonable.

Due to the inherent limitation of model checking approaches, we intend to use theorem proving techniques as the underlying verification method. This means providing theories for establishing the impossibility of particular combinations of events. These verification methods should be implemented in a feasible and sound environment. For this purpose, we intend to use first-order and higher-order logics based theorem proving techniques provided by the event-B Click'n'Prove tool [4] PVS tool [66], respectively.

In order to reduce the complexity of the verification method, abstraction techniques should be applied on the model. This, definitely will affect our certainty about the correctness of the property. While trying to prove a specific property for the protocol, we need to understand if we should consider all its primitives, or we can consider some of them irrelevant to that property. Similarly, we can consider the abstraction of the protocol environment, the number of protocol sessions and the number of protocol participants. This provides a new model for the protocol at higher level of abstraction, which is called *abstract protocol* [71]. This abstraction can serve in the design of protocols as well as their verification. It is an improvement on the model and the verification approach.

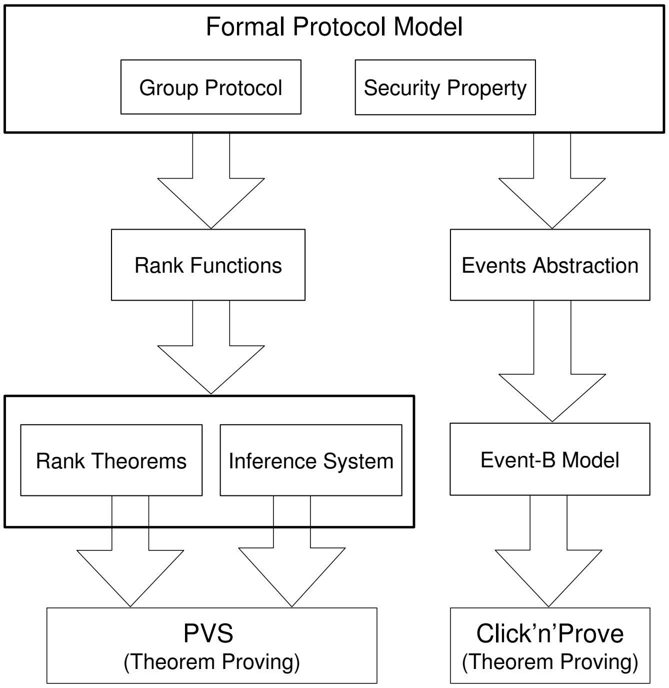


Figure 1.1: Verification Methodology

Figure 1.1 displays a general view of the proposed verification methodology. Three major techniques are used: in the first one, we use *rank theorems* to define the required security property based on the concept of rank functions. Rank theorem are implemented in PVS higher-order logic theorem prover. In the second approach, we provide a sound and complete *inference system* to detect attacks in group key protocols. The inference system is also implemented in PVS. Finally, we abstract our group protocol model to fit it to the first-order logic theorem proving in *event-B*. We define a well-formed formal link between the group protocol model and the event-B counterpart model. Based on this link, we propose a solution to verify the required security properties, in particular group key secrecy and forward secrecy properties, using the event-B invariant checking. The detailed view of each of the three methods is described below.

An overview of the rank theorems based approach is given in Figure 1.2, where a set of rank functions is defined, and theorems to prove their soundness are established. This set of rank functions is used in two directions: rank theorems and rank functions based inference system. A general rank theorem is established based on rank functions, and its correctness is established w.r.t the group protocol model and its security property. Then two theorems are established for forward and backward secrecy, and their correctness is established in PVS theorem prover. On top of forward and backward secrecy theorems, a rank theorem is defined and verified for collusion property in the same way. Note that the term "rank theorem" was first mentioned in [29] to refer to CSP theorems that encodes rank functions to model authentication in security protocols.

We apply the implemented proof environment on the Enclaves protocol from SRI [33] in order to verify related forward and backward secrecy properties. Then we defined a rank theorem for collusion property on top of forward and backward secrecy. In our approach, we establish the proof at the protocol level of abstraction, under the assumption of perfect cryptography and algebraic conditions. More details will be described in a dedicated chapter in the thesis.

The rank theorem is efficient in reasoning about the *absence* of attack, however, it is necessary to be able to reason about the *existence* of attacks too. In addition, implementing rank theorems requires a lot of user interaction with the verification tool because of the separation of rank functions from protocol events. This motivated us to improve the efficiency of this method, and at the same time provide a way to reason about existence of attacks.

Based on the set of sound rank functions, we propose a sound and complete inference system to detect attacks in group key protocols (see Figure 1.3). The inference system

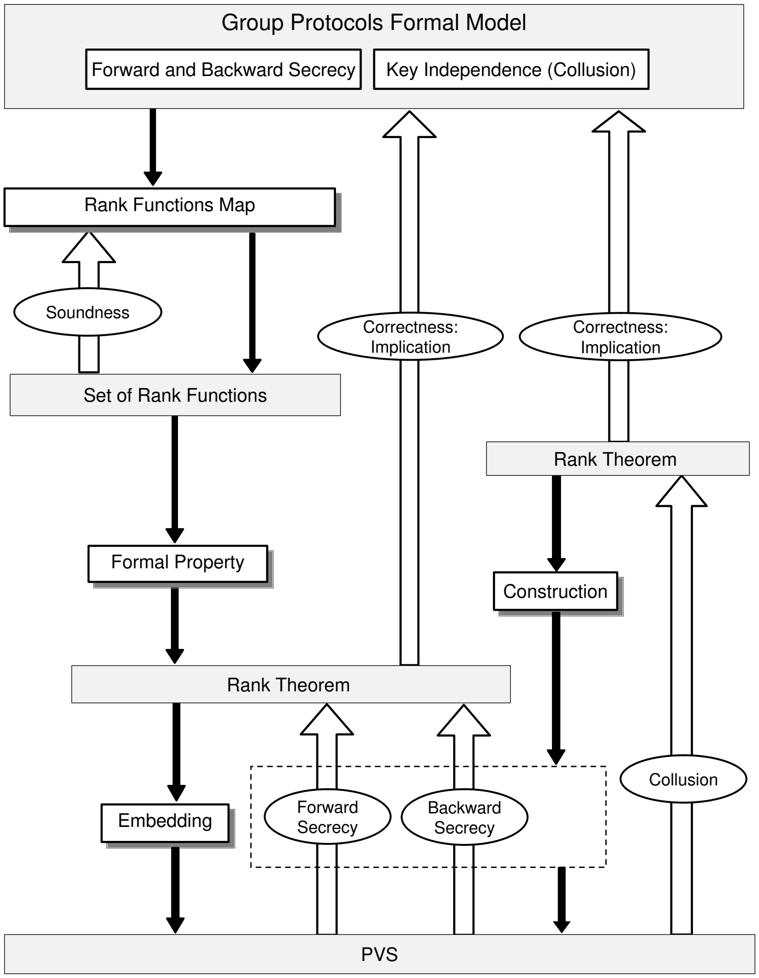


Figure 1.2: Rank Theorems based Approach

provides an elegant and natural proof strategy for such protocols compared to existing approaches. Rank functions are combined with protocol events in a set of inference rules to form a rank functions based inference system. Two theorems are established and proved for the soundness and completeness of the inference system w.r.t the protocol model and its security property. In this approach, it is necessary to show a formal link between the original security property and the verified one in the tool. The above formalizations and rank theorems were implemented using PVS. The inference system is implemented in PVS and applied on the Group Diffie-Hellman (GDH) protocol. This protocol is vulnerable to a known attack in the existence of an active adversary. Therefore, it is more appropriate than the Enclaves protocol in order to demonstrate the efficiency of this method in reasoning about the existence of attacks. More details will be described in a dedicated chapter in the thesis.

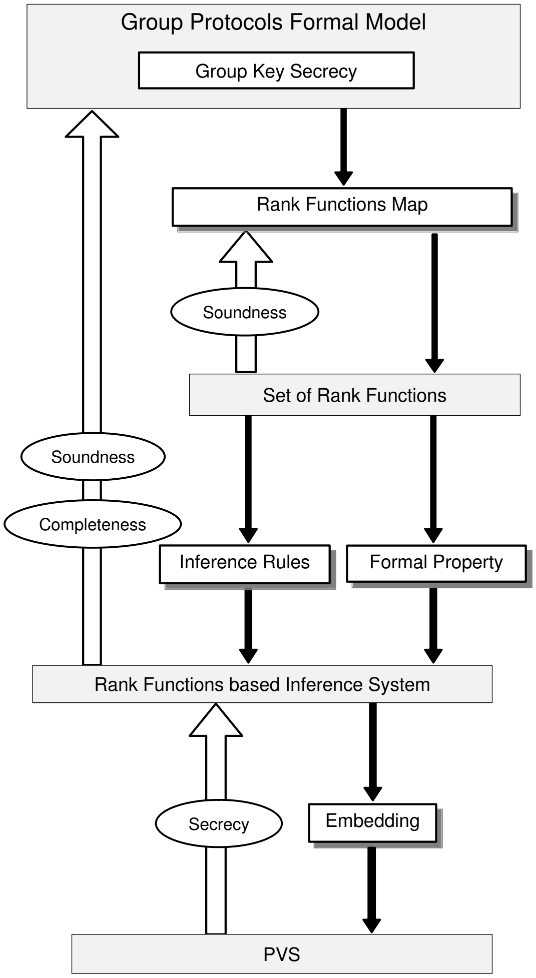


Figure 1.3: Inference System based Approach

Implementing the rank theorems and the inference system in higher-order logics theorem proving required a lot of effort and time, in addition, verifying properties is achieved interactively with the theorem proving tool because of the decidability problem of higher-order logics. In order to complement the previous methods, an event-B based automatic invariant checking is provided for a similar class of properties. This allows us to avoid user interaction with the theorem proving tool, and reduce the time required to verify such property. In this method, once a protocol is proven to be secure in the static case, it can be easily proven to be secure in the dynamic case by considering a protocol event, where, in this case, the number of members and messages in the protocol are abstracted under the execution of a single event.

In order to reason about group protocols in the first-order logics, a map between the group protocol model and event-B model semantics is defined. The event-B tool guarantees the correctness of the invariant w.r.t the event-B model. The map from group protocols to event-B model guarantees certain equivalence between the two models, under certain conditions. Secrecy property is semantically implied in event-B invariant in a defined and proved lemma. Then, a theorem is defined to guarantees that once an event-B invariant is proved against event-B mode, we can conclude that the secrecy property is correct for the group protocol mode. However, in the event-B method, compared to higher-order logics, the number of protocol participants that can considered is limited, it is also applicability on a single protocol event, finally, it cannot model backward secrecy and key independence.

We applied this method on the Tree-based Group Diffie-Hellman (TGDH) protocol [49], for which we have proven secrecy property based on event-B refinement and the Click'n'Prove tool. The TGDH protocol has a simple key management scheme based on binary-tree structure, therefore, it is suitable for this method, since it is based on the first-order logic, which is less expressive than higher order logic. Figure 1.4 depicts the formal links in the proposed event-B approach. More details will be described in a dedicated chapter in the thesis.

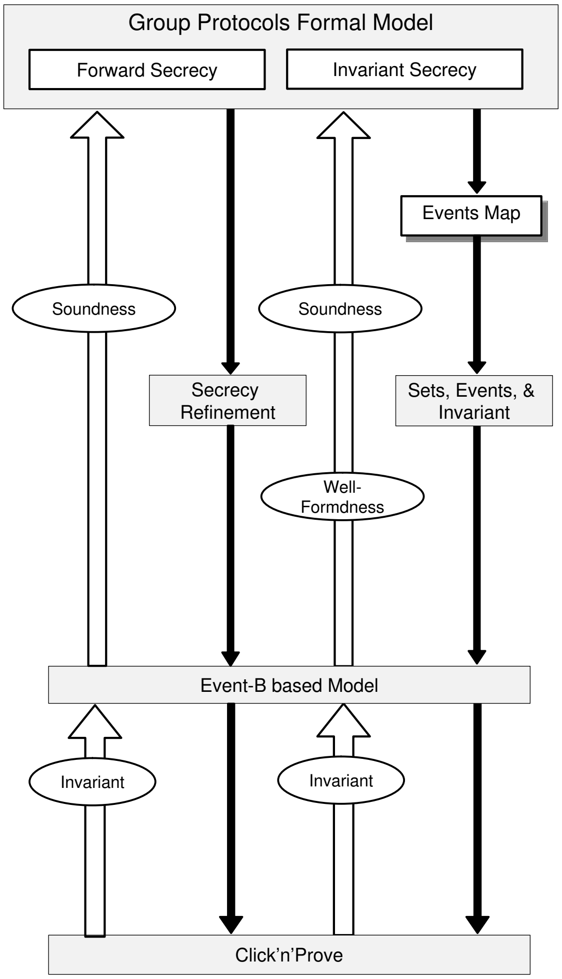


Figure 1.4: Event-B based Approach

1.5 Thesis Contributions

The primary focus of this thesis is on the idea of using higher-order and first-order logic theorem proving for the verification of security properties for group key protocols. The thesis makes the following contributions:

- Provide a set of generic requirements of group key protocols, establish their formal specifications, and define a formal model for this class of protocols. [Bio-CF-3,Bio-TR-5][1]

[1] The references with prefix Bio are provided in the Biography section at the end of the thesis.

- Verify security properties such as forward and backward secrecy, and, in addition, mechanically verify collusion property under certain assumptions. This is, to the best of our knowledge, the first such mechanized proof in the context of theorem proving [Bio-CF-2].

- Present rank theorems to enable and mechanize the verification procedure of this class of protocols, and prove the soundness of our approach by proving the correctness of the rank theorem. We implemented this approach in PVS and illustrated it on the Enclaves group protocol. [Bio-CF-3,Bio-TR-4]

- Provide a complete and sound inference system defined over rank functions. The approach is based on an elegant and natural proof strategy for the verification of group key protocols. We implement the inference system in PVS, and illustrate it on the Group Diffie-Hellman protocol. [Bio-JR-1,Bio-TR-3]

- Provide an automatic approach using first-order logic theorem proving system in the context of group key protocols verification to verify secrecy properties using event-B invariant checking. Then, present a method for modeling and verification of forward secrecy using event-B refinement. Finally, apply this method on the Tree-based Group Diffie-Hellman protocol. [Bio-CR-1,Bio-TR-1,Bio-TR-2]

1.6 Thesis Organization

The rest of the thesis is organized as follows.

In Chapter 2, we provide a brief introduction to the basic preliminaries and notations that will be used throughout the thesis. This includes group key protocols, protocol notations, protocol attacks, theorem proving techniques, the PVS theorem prover, event-B method, and the Click'n'Prove tool.

Chapter 3 describes a formal model for the protocol. First, we give the formal notations that will be used in the thesis, then, we give the formal specifications of secrecy properties and protocol events. Finally, we give the definition of rank functions and the requirements for a set of sound rank functions.

In Chapter 4, rank theorems for forward and backward secrecy properties are presented based on the group protocol model. We illustrate our approach on the verification of forward and backward secrecy for the Enclaves protocol using PVS. Then we show how to construct and prove collusion property from forward and backward secrecy.

Next, in Chapter 5, we introduce the inference system which is defined over a set of sound rank functions. The inference system is embedded in PVS theorem prover. We show an illustrative application of this system on a group key protocol, the Group Diffie-Hellman protocol, providing a mechanized approach using theorem proving in the context of group key protocols verification.

Chapter 6 presents an abstract approach for modeling and verification of group key protocols by using event-B first-order logic invariant checking and refinement checking. We present theorems that shows the correctness of the semantical link from the group protocol model to the counterpart event-B model. Then we apply the method on the tree based Group Diffie-Hellman protocol and verify it in the Click'n'Prove tool.

Finally, Chapter 7 concludes the thesis and outlines some future research directions.

Chapter 2

Preliminaries

In this chapter we provide an introduction to the preliminaries that will be used in the rest of this thesis. In the first section we introduce security protocols focusing more on group key protocols. We also show an example of group key protocols, and an attack example on group key protocols that can be generated taking advantage of the protocol messages interaction. Then, in the second section, we overview the concept of theorem proving approach and the PVS theorem proving tool. Finally, the notion of event-B method and the Click'n'Prove invariant checking tool are presented.

2.1 Group Key Protocols and Attacks

Cryptographic protocols are used for establishing secure communication in order to allow group members to exchange or establish keys to encrypt and authenticate messages within the group. In a group communications context, new members can join or leave the group, therefore the group requires that membership changes cause the group key to be refreshed. It is intuitive to generate a new key so that a new member is prevented from decoding messages exchanged before he/she joined the group. If a new key is distributed to the group when a new member joins, the new member cannot decrypt previous messages encrypted with the old key even if he/she has recorded any combination of earlier messages. In addition, changing the group key prevents a leaving group member from accessing the group communication if it can receive the messages. If the key is changed as soon as a member leaves, that member will not be able to decipher group messages encrypted with the new key. However, distributing the group key to valid members is a complex problem. Rekeying is a trivial method for sending the new group key to the old group members encrypted with the old group key. Although rekeying the group before a new member joins is trivial, rekeying the group after a member leaves is far more complex since any old key cannot be used to distribute a new one. Therefore, a scalable and reliable mechanism should be provided to generate and distribute the group key. Rafaeli and Hutchison [74] classified group key management into three different types:

- Centralized group key management protocols. A single entity is employed for controlling the whole group, hence a group key management protocol seeks to minimize storage requirements, computational power on both client and server sides, and bandwidth utilization.

- Decentralized architectures. The management of a large group is divided among subgroup managers, trying to minimize the problem of concentrating the work in a single place.

- Distributed key management protocols. There is no explicit key distribution center. Members can generate key and perform access control and the generation of the key can be either contributory, meaning that all members contribute some information to generate the group key, or done by one of the members.

2.1.1 Protocol Notation

A protocol is formulated as a sequence of messages, together with the names of the sender and receivers, a message is given as

Message $n \quad a \rightarrow b : data$

where, a and b are users, $data$ is the message content which can be composed of:

atoms: This may be names, variables and literal constants.

nonces: A nonce, usually notated like n_A, is an unpredictable, freshly generated unique number. The subscript is just a notational convenience indicating which participant created it, in the real protocol there is no attached name tag or something similar that indicates its creator.

encryption: The term $\{data\}_k$ denotes the encryption of data with the key k.

authentication: $Sign_k(data)$ denotes the signature of data using the key k.

concatenation: $a.b$ denotes the concatenation of a and b, i. e. the two terms are sent consecutively.

In the following, we show a simple protocol to illustrate these concepts, the protocol is a Challenge-Response, which can verify that two parties A and B share a common secret key k without revealing it. It is commonly employed after a key exchange to assure that the keys were not modified either accidentally or by an attacker:

1. $A \rightarrow B : \quad n_A$
2. $B \rightarrow A : \quad \{n_A\}_k.n_B$
3. $A \rightarrow B : \quad \{n_B\}_k$

After receiving the first message, B encrypts the nonce N_A with his/her version of k and sends it back. Now A can decrypt it again and compare the result to the number he/she originally sent. If they match, then under the assumption that k is not known to any attacker, A can be sure that B has the same k as he/she has. The challenge is then performed the other way around for convincing B of the fact.

2.1.2 Group Key Protocols Example

An example on group key protocols is the Diffie-Hellman key exchange [32]. This protocol, also called exponential key exchange, enables two participants to calculate a common symmetric secret key solely from public information. It is based on public-key cryptography and provides a practical solution to the key distribution problem. It enables two parties, which have never communicated before, to establish a shared secret key by exchanging messages over a public channel.

A prime number p and a primitive element g, where the powers of g run through all possible remainders modulo p, are publicly known and can be the same for all participants. The protocol participants A and B randomly choose their own secret keys a, and b respectively. The values g^a and g^b can be published as public keys. Now the participants can calculate the shared secret key $k = (g^a)^b = (g^b)^a = g^{ab}$.

1. $A \rightarrow B : \quad g^a$
2. $B \rightarrow A : \quad g^b$
3. $A\ computes\ k = (g^b)^a$
4. $B\ computes\ k = (g^a)^b$

The Computational Diffie-Hellman problem states that given $<\ g,\ g^a,\ g^b\ >$ it is hard to compute g^{ab}. In addition the Decision Diffie-Hellman Problem states that given

$< g,\ g^a,\ g^b,\ g^c >$, it is hard to check if $g^c = g^{ab}$. Diffie-Hellman based protocols presented in the literature rely on the complicity of these algebraic problems.

There are many extended protocols in the literature that are based on the DH agreement. For instance, Steiner *et al.* [78] presented a Diffie-Hellman key distribution extended to group communication in three different versions GDH.1, GDH.2, and GDH.3. They extended the well-known DiffieHellman key exchange method to groups of n parties. The authors called refers to these protocols as *initial key agreement* (IKA) within a group. Once a group is formed and the secure key is agreed upon, new members may join and group members may leave. Any membership change must cause a corresponding group key change in order to preserve group secrecy. Instead of rerunning full IKA for each membership change, the authors suggested a less expensive solution called auxiliary key agreement (AKA) [79], these protocols are claimed to be secure based on the polynomial indistinguishability of a DiffieHellman key from an arbitrary random value.

Kim *et al.* [49] designed a static group key exchange protocol based on the extension of the basic DH protocol by Steiner *et al.* [78]. The protocol is called Tree-based Group Diffie-Hellman (TGDH) and it outputs group keys using the logical structure of a balanced binary tree. The Internet Key Exchange (IKE) [41], and its latest version IKEv1 [43] are also based on the basic Diffie-Hellman protocol.

2.1.3 Attacks on Protocols: Example

Man-in-the middle attack is a “form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection when in fact the entire conversation is controlled by the attacker. The attacker must be able to intercept all messages going between the two victims and inject new ones, which is straightforward in many circumstances” [36].

An example of such an attack exists in the Diffie-Hellman key establishment scheme. Despite the fact that the protocol is designed to work in the existence of passive adversary, it is vulnerable to this attack in the existence of active adversary.

To execute the protocol, we chose a prime p and a primitive root $g\ mod\ p$. Primitive is an algebraic term that means all numbers between 1 and p can be generated by taking exponents of $r\ mod\ p$. The protocol works as follows, A chooses at random an integer a and sends B the message:

$A \rightarrow B: \quad m_1 = g^a\, mod\, p$

B chooses an integer b and sends A the message

$B \rightarrow A: \quad m_2 = g^b\, mod\, p$

A calculates

$A\ computes\ k_1 = g^{b^a}\ mod\, p$

B calculates

$B\ computes\ k_2 = g^{a^b}\ mod\, p$

It is easy to prove that $k_1 = k_2$. Hence A and B can use K_1 as a private key between themselves. Note that A and B play a symmetric role in the generation of the key. Deriving a from m_1 and b from m_2 is considered hard problem [78].

This Diffie-Hellman scheme has no way to ensure authentication. A man-in-the-middle could pretend to be B and establish a shared key with A, therefore, reading all messages that A believes to be sending to B. Similarly, the intruder could be doing the same with B, with certain control over the network. The intruder monitors a run of the protocol or part of it and at some time replays one or more of the messages. The intruder tricks an agent into revealing some information, possibly by inducing him to perform specific steps of the protocol.

2.2 Theorem Proving and PVS

Formal methods allow a thorough analysis of the different paths which an intruder can take, and also to specify precisely the system's boundary, i.e., the interface between the system and its environment. They can characterize a system's behavior and its desired properties precisely. They can prove that a system meets its specification and tell us under what circumstances a system does not meet its specification by providing counterexamples like intruder scenarios. Formal methods also help the designer to think about the protocol in a proper and thorough way since he/she must have a clear idea of what exactly he/she wants to achieve and must make assumptions explicit and non-ambiguous.

It should be emphasized that any proof of correctness is relative to both the formal specification of the system and the formal specification of the desired properties. A system proven correct with respect to an incorrect specification leaves us with no assurance about the system at all. Another problem is the difference between an abstract mathematical model and a real-world instantiation of a system. Systems do not run isolated; they operate in some environment. The formal specification of a system must always include the assumptions made about an environment. As soon as one of these assumptions does not hold, the conclusion is invalid. The main formal verification techniques are: model checking and theorem proving.

One of the earliest approaches to formal verification was to describe both the implementation as well as the specification in a formal logic. The correctness result was then obtained by proving in the logic, that the specification and implementation were suitably related. Theorem proving is a technique where both the system and its desired properties are expressed as formulas in some mathematical logic. The logic is given by a formal system based on a set of axioms and inference rules. Theorem proving is the process of finding proofs using axioms of the system. Steps in the proof appeal to the axioms and rules, and possibly derived definitions and intermediate lemmas. Theorem provers based on higher-order logic are distinguished by a high level of expressiveness and generality compared to model checkers. They do, however, require a lot of user expertise and interactivity. Although, theorem proving is a slow process that requires much of human interaction, it gives invaluable insight to the user on the system or the property being proved.

The powerful mathematical techniques such as abstraction and induction are strong points of theorem proving. This make it a very flexible and powerful verification technique. Formal logics that support theorem proving make it possible to construct a model at different levels of abstraction and to prove properties on all classes of systems. However, it is a time consuming process which can involve generating and proving literally hundreds of lemmas.

Higher-order logic [17] is derived from the typed λ calculus. This logic is more powerful than first or second-order. First-order logic [39] can only quantify over individual variables, for instance, $\forall x, y R(x, y) \Rightarrow P(x, y)$. Second-order logic can quantify over predicates and functions, for instance, $P \Rightarrow Q \equiv \forall R.(P \Rightarrow Q \Rightarrow R) \Rightarrow R$; whereas higher-order logic can quantify over arbitrary functions and predicates. Any proposition of first-order logic can be translated into a proposition of higher-order logic, but the reverse does not hold. However, higher-order logic has, some disadvantages. First it is incomplete, second it is undecidable. Undecidable means there is no effective method for determining whether arbitrary formulas are theorems of the logical system. Completeness means that there are no true sentences in the system that cannot be proven in the system. Also there is no complete deduction system for the second-order logic. Reasoning is more difficult in higher order than in first-order logic. In addition, for any proof system, inference rules and heuristics are needed. Finally, inconsistencies can arise in higher-order systems if semantics not carefully defined [50].

Theorem proving is a formal verification method in which the correctness of a design is formulated as a theorem in a mathematical logic and the proof is checked using a general-purpose theorem-prover. Based on first-order and high-order logic.

The verification of a theorem depends on the underlying logic used in theorem proving. First-order logic is semi-decidable because it is restricted to propositional calculus and terms. Therefore sound and complete first-order logic automated reasoners are available and can conduct completely automated proofs. On the other hand, the more expressive logics, second and higher-order logics, can be used to model a wider range of problems than first-order logic, however, theorem proving cannot be fully automated and requires user interaction to guide the proof tools.

Theorem proving methods have been used for the verification of wide range of systems: hardware, software, protocols, hybrid, and many others. Among the mostly used theorem provers are HOL (Higher-Order Logic) [40], Isabelle [67], PVS (Prototype Verification System) [66], Coq [23], ACL2 [47], and Click'n'Prove [4]. The major differences between these systems is the way automatic decision procedures are integrated into the system. This results in a different degree of automation among each others. In addition, these systems use slightly different mathematical logics. However, they all require expertise in using theorem proving technique, which is not fully automated and requires a large amount of time to verify systems. The advantage of the deductive verification approach is its ability to handle and verify complex systems because of the expressiveness of the logic used in theorem provers. In this, we will use the PVS higher-order logic theorem prover and the Click'n'Prove first order theorem prover. In the next section, we will briefly overview the PVS theorem proving system.

2.2.1 Prototype Verification System (PVS)

The Prototype Verification System (PVS) [66, 72, 76] is a mechanized environment for formal specification and verification of systems. “PVS consists of a specification language, a number of predefined theories, a type checker, formalized libraries, and an interactive theorem prover that supports the use of several decision procedures and a symbolic model checker with various utilities including a code generator, a random tester, and documentation” [72].

The specification language of PVS is based on classical, typed higher-order logic. The base types include uninterpreted types that may be introduced by the user, and built-in types such as the Booleans, integers, and reals. The type-constructors include functions, sets, tuples, records, enumerations, and inductively-defined abstract data types, such as lists and binary trees, and abstract data types. Predicate subtypes and dependent types can be used to introduce constraints over user defined types as Type-Correctness Conditions (TCCs), a separate collection of lemmata that must be proven to ensure well-typedness. PVS specifications are organized into parameterized theories that may contain assumptions, definitions, axioms, and theorems. Parameters can include constants, types, and theories. Theory interpretations are used to instantiate the declared types and constants in a theory with definitions. PVS expressions provide the usual arithmetic and logical operators, function application, lambda abstraction, and quantifiers, within a natural syntax [72].

The PVS theorem prover presents the current proof goal in the form of a sequent [76]:

```
[ -1] A1
[ -2] A2
[ -3] A3
        ⋮
     |-------
```

```
[1] B1
[2] B2
      ⋮
```

A_i and B_i are referred to as *sequent formulas* and consists of a number of antecedents (Ai) and consequents (B_j). Note that antecedents or consequents can be empty, but not both. The sequent intuitively means that the conjunction of the antecedents should imply the disjunction of the consequents. The sequent represents the proof goal: $A1 \wedge A2 \wedge A3 \Rightarrow B1 \vee B2$, where $A1, ..., B2$ are propositions in the PVS language.

PVS expressions provide the usual arithmetic and logical operators, function application, lambda abstraction, and quantifiers, within a natural syntax. Names may be freely overloaded, including those of the built-in operators such as AND and +. Tabular specifications are supported, with automated checks for disjointness and coverage of conditions. An extensive prelude of built-in theories provides hundreds of useful definitions and lemmas.

The PVS language provides interpreted or uninterpreted type declarations. In addition, subtype declarations are supported.

uninterpreted type : `T: TYPE`

uninterpreted subtype : `S: TYPE FROM T`

interpreted type : `T: TYPE = int`

enumeration type : `T: TYPE = {r, g, b}`

Variable declarations introduce new variables and associate a type with them. These are logical variables, not program variables; they simply provide a name and associated type so that binding expressions and formulas can be succinct. Constant declarations introduce new constants, specifying their type and optionally providing a value. Since PVS is a higher order logic, the term constant refers to functions and relations.

```
n: int
```

```
c: int = 3
f: [int -> int] = (lambda (x: int): x + 1)
g(x:int): int = x + 1
```

The declaration for n simply introduces a new integer constant. Nothing is known about this constant other than its type, unless further properties are provided by `AXIOMs`. The other three constants are interpreted.

PVS allows a restricted form of recursive definitions for total functions so that the function is defined for every value of its domain. This is ensured by a `measure`, which is a function whose signature matches that of the recursive function. Recursive definitions are treated as constant declarations, except that the defining expression is required, and a measure must be provided.

Formula declarations introduce axioms, assumptions, theorems, and obligations. The expression that makes up the body of the formula is a boolean expression. Theorems may be introduced with many keywords such as `CHALLENGE`, `COROLLARY`, `FORMULA`, `LEMMA`, `PROPOSITION`, `SUBLEMMA`, or `THEOREM`.

The PVS theorem prover provides a collection of powerful primitive inference procedures that are applied interactively under user guidance within a sequent calculus framework. The primitive inferences include propositional and quantifier rules, induction, rewriting, simplification using decision procedures for equality and linear arithmetic, data and predicate abstraction, and symbolic model checking. The implementations of these primitive inferences are optimized for large proofs: for example, propositional simplification uses BDDs, and auto-rewrites are cached for efficiency. User-defined procedures can combine these primitive inferences to yield higher-level proof strategies.

Proof commands can be used by the user to introduce lemmas, expand definitions, eliminate quantifiers (*skolemise*), split up a disjunct, or make a case distinction. More complex commands are also available, such as apply decision procedures, rewrite the formula, apply a certain lemma, simplify using decision procedures, or even user-defined proof strategies. Proofs yield scripts that can be edited, attached to additional formulas, and rerun. This allows many similar theorems to be proved efficiently, permits proofs to be adjusted economically to follow changes in requirements or design, and encourages the development of readable proofs [72].

2.3 Event-B Method

The event-B method [3] uses the set-theoretical and logical notations of the B method and provides new notations for expressing abstract models based on events. It provides invariants proofs for a state-based system that is updated by guarded events. In order to model a group key protocol in event-B first order logic, the semantics of the event-B language should be formally related to the protocol model.

In event-B, an event consists of a guard and an action. The guard is a predicate built on state variables and the action is a generalized substitution which defines a state transition. An event may be activated once its guard evaluates to true and a single event may be evaluated at once. The system is assumed to be closed and it means that every possible change over state variables is defined by transitions; transitions correspond to events defined in the model. The B method is based on the concept of machines (or systems) [2]. A machine is composed of descriptive and operational specification:

SYSTEM $< name >$

SETS $< sets >$

VARIABLES $< variables >$

INVARIANT $< invariants >$

INITIALISATION $< initialization\ of\ variables >$

EVENTS $< events >$

END

A descriptive specification describes what the system does by using a set variables, constants, properties over constants and invariants which specify properties that machine's state verify. This constitutes the static definition of the model.

Operational specifications describe the way the system operates. It is composed of a set of atomic events described by generalized substitutions. Formally, several substitutions are defined in B. If we consider a substitution S and a predicate P representing a post-condition, then $[S]P$ represents the weakest precondition that establishes P after execution of S. The substitutions occurring in event-B models are inductively defined by the following expressions:

[**SKIP**]$P \Leftrightarrow P$

[**BEGIN** S **END**]$P \Leftrightarrow [S]P$

$[S1||S2]P \Leftrightarrow [S1]P \wedge [S2]P$

[**ANY** v **WHERE** G **THEN** S **END**]$P \Leftrightarrow \quad v[P \Rightarrow [S]P]$

[**SELECT** G **THEN** S **END**]$P \Leftrightarrow G \Rightarrow [S]P$

$[x := E]P \Leftrightarrow P(x/E)$

$P(x/E)$ represents the predicate P where all the free occurrences of x are replaced by the expression E. The guard and the action of an event define a before-after predicate for this event. It describes relations between variables before the event holds and after this. Proof obligations are produced from events in order to state that the invariant condition is preserved. Let M be an event-B model with v being variables, carrier sets or constants. The properties of constants are denoted by $P(v)$, which are predicates over constants, and the invariant by $I(v)$. Let E be an event of M with guard $G(v)$ and before-after predicate $R(v, v')$. The initialization event is a generalized substitution of the form $v : init(v)$. Initial proof obligation guarantees that the initialization of the machine must satisfy its invariant: $Init(x) \Rightarrow I(x)$.

An event has a guard and an action, and it may occur only when its guard evaluates to true. An event has one of the general forms where the $SELECT$ form is just a particular case of the ANY form.

2.3.1 Invariant

The consistency of an event-B model is established by proof obligations which guarantee that the initialization verify the invariant and that each event should preserve the invariant. The guard and the action of an event define a before-after predicate for this event. It describes a relation between variables before the event holds and after this. Proof obligations are produced from events in order to state that the invariant condition is preserved.

Each event E, if it holds, has to preserve its invariant. The feasibility statement is illustrated in Lemma 2.3.1 and the invariant preservation is given in Lemma 2.3.2 [61].

Lemma 2.3.1 $I(v) \ \wedge \ G(v) \ \wedge \ P(v) \Rightarrow \exists v'.R(v, \ v')$

Lemma 2.3.2 $I(v) \ \wedge \ G(v) \ \wedge \ P(v) \ \wedge \ R(v, v') \Rightarrow \ I(v')$

An event-B model M with invariants I is well-formed, dented by $M \models I$ only if M satisfies all proof obligations. The B syntax for generalized substitutions defines three predicates: a relation R, the subsets of the pre-states where G is true of the states in $domain(R)$, and the subset of the pre-state where P is true. Let S be restricted to evaluations that satisfy the invariant, $S \triangleq \{v|I(v)\}$. Each event can be represented by a binary relation rel. rel is formally defined as $rel \triangleq \{v \mapsto v' \mid I(v) \ \wedge \ G(v) \ \wedge \ R(v,\ v')\}$. The fact that the invariant $I(v)$ is preserved by event rel is simply formalized by saying that rel is a binary relation built on S: $rel \subseteq S \times S$. It is shown that this binary relation yields to both Lemmas 2.3.1 and 2.3.2 above [61].

The B syntax for generalized substitutions defines three predicates: a relation R and, the subsets of the pre-states where G is true of the states in $domain(R)$, and the subset of the pre-state where P is true. Let S be restricted to evaluations that satisfy the invariant, $S \triangleq \{v|I(v)\}$. Each event can be represented by a binary relation rel. rel is formally defined as $rel \triangleq \{v \mapsto v' \mid I(v) \ \wedge \ G(v) \ \wedge \ R(v,\ v')\}$. The fact that the invariant $I(v)$ is preserved by event rel is simply formalized by saying that rel is a binary relation built on S: $rel \subseteq S \times S$. It is shown that this binary relation yields to both Lemmas 2.3.1 and 2.3.2 above [61].

Lemma 2.3.1 guarantees that the active part of the relation is a total relation, i.e., when all predicates I, P, and G hold, formally $G(v) \wedge P(v) \subseteq domain(R(v,\ v'))$. Finally, Lemma 2.3.2 guarantees that the postcondition of any operation must satisfy the machine invariant. Initial proof obligation guarantees that the initialization of a machine must satisfy its invariant.

Special rules for the initialization events are distinguished. $R_I(v, v')$ is used to denote the predicate of the generalized substitution associated with this event. The following initialization statements is obtained [61]:

Lemma 2.3.3 $P(v) \Rightarrow \exists v'.R_I(v, v')$

Lemma 2.3.4 $P(v) \ \wedge \ R_I(v,\ v') \Rightarrow \ I(v')$

2.3.2 Refinement

Refinement is a technique to deal with the development of complex systems. It consists in building, starting from an abstract model, a sequence of models of increasing complexity containing more and more details. These details could be introduced when using new variables, adding details to abstract events or adding new events. A model in the sequence is followed by a model it refines. The invariant of the refined model is not weaker than the model it refines and it may contain new variables. The events are the same but may be redefined. It is also used to transform an abstract model into a more concrete version by modifying the state description [3]. The abstract state variables, v, and the concrete ones, v_c, are linked together by means of a gluing invariant $J(v, v_c)$. A number of proof obligations ensures that (1) each abstract event is correctly refined by its corresponding concrete version, (2) each new event refines skip, (3) no new event takes control forever, and (4) relative deadlock fairness is preserved. Suppose that an abstract model M with variables v and invariant $I(v)$ is refined by a concrete model M_c with variables v_c and gluing invariant $J(v, v_c)$. If $R_A(v, v')$ and $R_C(v_c, v'_c)$ are respectively the abstract and concrete before-after predicates of the same event, we have to prove the following statement:

$$(I(v) \land J(v, v_c) \land R_C(v_c, v'_c)) \Rightarrow \exists v'.(R_A(v, v') \land J(v', v'_c))$$

This statement means that under the abstract invariant $I(v)$ and the gluing invariant $J(v, v_c)$, a concrete step $R_C(v_c, v'_c)$ can be simulated (v') by an abstract one $R_A(v, v')$ in such a way that the gluing invariant $J(v', v'_c)$ is preserved. A new event with before-after predicate $R(v_c, v'_c)$ must refine skip $(x' = x)$. This leads to the following statement to prove : $I(v) \land J(v, v_c) \land R_C(v, v'_c) \Rightarrow J(v, v'_c)$. Moreover, we must prove that a variant $V(v_c)$ (valuation of variable v) is decreased by each new event (this is to guarantee that an abstract step may occur). We have thus to prove the following for each new event with before-after predicate $R_c(v_c, v'_c)$:

$I(v) \wedge J(v, vc) \wedge BA(v, v') \Rightarrow Val(v_c') < Val(v_c)$. At last, we must prove that a concrete model does not introduce more deadlocks than the abstract one. This is formalized by means of the following proof obligation:

$I(v) \wedge J(v, v_c) \wedge G(M) \Rightarrow G(M_c)$

where $G(M)$ stands for the disjunction of the guards of the events of the abstract model, and $G(M_c)$ stands for the disjunction of the guards of the events of the concrete one. The essence of the refinement relationship is that it preserves already proved system properties including safety properties. The invariant of an abstract model plays a central role for deriving safety properties; the goal is to obtain a formal statement of properties through the final invariant of the last refined abstract model.

2.3.3 Click'n'Prove

Click'n'Prove [4] is a predicate theorem prover which offers an automatic semi-decision procedure for first-order logics, and a systematic translation of statements written within set theory into equivalent ones in first-order predicate calculus.

The proof assistant deals with the proofs of sequents of the following form:

$hypothesis \vdash conclusion$

where $hypothesis$ denotes a possibly empty list of predicates and $conclusion$ denotes a predicate which is not conjunctive and usually implicative. Conducting a proof within this framework leads to the application of some pre-defined inference rules able to transform such a sequent into zero, one or more successor sequents.

In the Click'n'Prove language, the key words $constants$ and $variables$ are used to define corresponding terms as follows:

```
VARIABLES
   x
   , y
  , m : 0..N
```

```
CONSTANTS
    a
    , b
```

An invariant is a condition on the state variables that must hold permanently. In order to achieve this, it is just required to prove that, under the invariant in question and under the guard of each event, the invariant still holds after being modified according to the transition associated with that event. Invariants are defined as follows:

```
INVARIANT
   w : BOOL
 & t : BOOL
 & (x = FALSE  & t = FALSE => m = N)
```

Initializations are defined similarly as follows:

```
INITIALISATION
    m := 0
 || w := FALSE
 || t := TRUE
```

Events are defined using $SELECT$ or ANY statement described above:

```
EVENTS
```

```
 Event1 = skip;
 Event2 =
   SELECT
      w = FALSE
    & m /= N
   THEN
      t := TRUE
   END
;
```

Chapter 3

Formal Model for Group Key Protocols

In this Chapter, we introduce our formal specification requirements for group protocols upon which we build each of the proposed verification methods. The intent is to introduce the basic concepts notations that are going to be used in the rest of the thesis. The formal notations to be used throughout the thesis is first presented. Then the formal definition of group secrecy, including forward secrecy, backward secrecy, and key independence is presented. Finally, the definition of rank function is presented, then the correction of a set of sound rank functions, w.r.t. specific requirements, is established.

The formal notations is going to be used in the rank theorems method (chapter 4), the inference system (chapter 5), and event-B based method (chapter 6). Rank functions will only be used in chapters 4 and 5.

3.1 Formal Notations

The formal definition of group protocols requirements is necessary first for understanding the protocol, and second for the protocol verification. Therefore, we are concerned with providing a formal definition for these requirements as a first step. We will consider different protocol design approaches, explore how these requirements are provided in these protocols, and formally model them accordingly. Of these approaches, we mention here the work of Wong *et al.* [85] about securely distributing re-key messages, Kikuchi [48] about a scheme for group key distribution, Kim *et al.* [49] about tree-key for group key management protocols, and finally the work of Rafaeli and Hutchison [73] about a decentralized architecture to create and distribute symmetric cryptographic keys.

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 protocols [59]. 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 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 this section we present our formal model and the notations we will use throughout this thesis.

$\mathbb{M}$: set of all possible messages (message space).

P: a honest principal who is willing to communicate.

$\mathbb{P}$: 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. They 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/her unlimited resources and computational power.

In this thesis, we consider an attacker-centric threat modeling. The model starts with an attacker, and evaluates their goals, and how they might achieve them. We state normal assumptions about the intruder such as being able to encrypt or decrypt a message only if he/she knows the appropriate key, or the ability to block, read, modify, or insert any message in the system.

$\mathbb{E}$: set of all events, or dynamic operations, i.e., join, leave, merge, and split. An event is a term from the message space to the message space, $\mathbb{E} : \mathbb{M} \rightarrow \mathbb{M}$. It represents an action the user can perform in order to obtain extra information and update his/her own set of knowledge.

attack: we define attack w.r.t. confidentiality as the ability of the intruder to have a message in the set of secret messages in his own set of knowledge, $attack \equiv m \in \mathbb{K}$ and $m \in \mathbb{S}$. The notion of attack can be seen differently depending on the nature of the security property under investigation.

$K_{\mathbb{G}_t}$: the group session key is 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. $I \notin \mathbb{G}_{t-i} \Rightarrow K_{\mathbb{G}_{t-i}} \in \mathbb{S}$.

Group membership $\in$: we define membership as follows: $K_{\mathbb{G}_t} \in \mathbb{P} \implies P \ \in \ \mathbb{G}_t$, 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 $\mathbb{P}$.

$\mathbb{T}$: 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 $\tau : \mathbb{E} \times \mathbb{M} \rightarrow \mathbb{M}$, $m \in \mathbb{M}$, then we write $m = \tau(\mathbb{E}, \mathbb{M})$ to say that a message m is generated by the trace τ by executing the set of events E on the set of messages M, we also write $\tau(\mathbb{E}, \mathbb{M}) \rightsquigarrow m$ to represent a predicate formula that evaluates to true if and only if $m = \tau(\mathbb{E}, \mathbb{M})$.

$\mathbb{K}_0$: 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 executing the protocol events. This information is usually public and known, so there are no secret information that is in the intruder's initial set of knowledge. In other words $\forall m \in \mathbb{M} : m \in \mathbb{S} \Rightarrow m \notin \mathbb{K}_0$

$\mathbb{K}$: set of knowledge of the intruder. The intruder updates this knowledge by executing events. The intruder starts with the initial set of knowledge and the set of events, then, by executing a sequence of events, he/she updates this set. $\mathbb{K}_0 \subseteq \mathbb{K}$ and $\mathbb{K} \subseteq \mathbb{M}$.

$\mathbb{K}_f$: set of knowledge of a user who is not currently a member of the group, i.e. has no access to the current group's secret shares, and who was previously a member of the group. $\mathbb{K}_f \cap \mathbb{K} = \emptyset$. Also $\mathbb{K}_f \subset \mathbb{S}$, where $\mathbb{S}$ is the set of secret messages when this user was member of the group. This is used to express the knowledge a member accumulated during activity as a member of the group, and it will be used to model forward secrecy.

$\mathbb{K}_b$: set of knowledge of a user who is not currently a member of the group, i.e. has no access the current group's secret shares, and who will be a member of the group in the future. $\mathbb{K}_b \cap \mathbb{K} = \emptyset$. Also $\mathbb{K}_b \subset \mathbb{S}$, where $\mathbb{S}$ is the set of secret messages when this user will be a member of the group. This is used to express the knowledge a member will accumulate during activity as a member of the group, and it will be used to model backward secrecy.

$\mathbb{G}_t$: 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.

3.2 Group Secrecy and Events

3.2.1 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 hence after. 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 3.2.1 ***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 $K_{\mathbb{G}_t}$. We use ϕ to represent group key secrecy property.*

$I \notin \mathbb{G}_t \Rightarrow \neg\exists\tau \in \mathbb{T}$: $K_{\mathbb{G}_t} = \tau(\mathbb{E}, \mathbb{M})$, where $\mathbb{E}$ is the set events and $\mathbb{M}$ is the set of messages.

This can be expressed as $\phi \equiv \mathbb{K} \cap \mathbb{S} = \emptyset$

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 3.2.2 ***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 $0 < i < t$.*

$I \in \mathbb{G}_t \Rightarrow \neg\exists\tau \in \mathbb{T} : K_{\mathbb{G}_{t-i}} = \tau(\mathbb{E}, \mathbb{M})$, where $I \notin \mathbb{G}_{t-i}$, and $0 < i < t$.

This can be expressed as $\phi_f \equiv (\mathbb{K} \cup \mathbb{K}_f) \cap \mathbb{S} = \emptyset$

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 3.2.3 ***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 $i > 0$.*

$I \in \mathbb{G}_t \Rightarrow \neg\exists\tau \in \mathbb{T} : K_{\mathbb{G}_{t+i}} = \tau(\mathbb{E}, \mathbb{M})$, where $I \notin \mathbb{G}_{t+i}$ and $i > 0$.

This can be expressed as $\phi_b \equiv (\mathbb{K} \cup \mathbb{K}_b) \cap \mathbb{S} = \emptyset$

Definition 3.2.4 ***Key Independence****: Key Independence guarantees that passive adversaries who know any proper subsets of group keys cannot collude their knowledge in order to discover any other secret group key.*

Key independence is known as collusion resistance property. The above definition is a generalized one. In our treatment, we restrict the definition into colluding the knowledge of two members only. This way, key independence can be defined in terms of forward and backward secrecy being satisfied simultaneously as follows:

$\phi_c \equiv (\mathbb{K}) \cup (\mathbb{K}_b \cup \mathbb{K}_f) \cap \mathbb{S} = \emptyset$

This means that colluding the knowledge of two members, one of them being the intruder, and using the new set of messages to update $\mathbb{K}$ should not result in any message in $\mathbb{S}$.

3.2.2 Joining and Leaving Groups

Any group key 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. In our model, we assume that generating and distributing the new key is done whenever a single member joins or leaves the group, therefore, we do not treat batch rekeying where a specific number of members should join or leave the group before the protocol generates and distributes the new key.

Definition 3.2.5 *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 $K_{\mathbb{G}_{t+i}} = \tau(\mathbb{E}, \mathbb{M})$ 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 3.2.6 *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 $P \notin \mathbb{G}_{t+i}$).*

3.2.3 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 3.2.7 *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 $K_{\mathbb{G}_{t+i}} = \tau(\mathbb{E}, \mathbb{M})$, $K_{\mathbb{G}_{t+i}} \neq K_{\mathbb{G}1_t}$ and $K_{\mathbb{G}_{t+i}} \neq K_{\mathbb{G}2_t}$.*

Definition 3.2.8 *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.

3.2.4 Traces

We define traces since it will be used to prove the soundness of the inference system. We use t to represent a single execution of one event or inference rule that updates the intruder set of knowledge. For a given events $e_1, e_2, ..., e_n \in \mathbb{E}$, the messages generated by executing each event: $m_1 = e_1(\mathbb{M}_0), m_2 = e_2(\mathbb{M}), ..., m_n = e_n(\mathbb{M})$. Now we can define $t_1, t_2, ..., t_n$ as follows :

$$t_1 : m_1 = e_1(\mathbb{K}_0),\ \rho(m_1) = c_1,\ \mathbb{K}_1 = \mathbb{K}_0 \cup \{m_1\}$$

$$t_2 : m_2 = e_2(\mathbb{K}_1),\ \rho(m_2) = c_2,\ \mathbb{K}_2 = \mathbb{K}_1 \cup \{m_2\}$$

$$t_i :\ m_i = e_i(\mathbb{K}_i),\ \rho(m_i) = c_i,\ \mathbb{K}_i = \mathbb{K}_{i-1} \cup \{m_i\}$$

$$\vdots$$

$$t_n : m_n = e_n(\mathbb{K}_n),\ \rho(m_n) = c_n,\ \mathbb{K} = \mathbb{K} \cup \{m_n\}$$

We define a trace $T_n \in \mathbb{T}$ as the sequence of executing $t_1, t_2, ..., t_n$ in order.

$T_n : t_1, t_2, ...t_n$; $\mathbb{K} = \mathbb{K}_0 \cup \{m_1, m_2, ..., m_n\}$; $\rho(m_n) = c_n$. We say that $m_n = T_n(\mathbb{E}, \mathbb{M})$ which means that the trace T_n generates the message m_n of rank c_n.

3.3 Rank Functions

A *rank function* is a map between the set of facts about the protocol and the set of natural integers. The set of facts include protocol events, protocol execution traces, keys, and messages. This map assigns a value or *rank* to each fact, such that facts that can be generated by the protocol 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. This map function will be useful in partitioning the message space and enabling mechanized proof of security protocols properties. The set of events and traces are concretely defined by the protocol, which allows defining them at different levels of abstraction in the final step of our approach.

The definition of the rank function is formally given as follows:

Definition 3.3.1 *Rank Functions [35, 75]. A rank function ρ is a map function $\rho : \mathbb{M} \rightarrow \mathbb{Z}$ which maps the set of all messages into integers.*

An appropriate rank function should be defined 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. In general, security properties are concerned with conditions under which the intruder can learn specific facts. We require that facts are obtained by protocol principals only be possible after some other fact (like authentication) has occurred. In order to establish a proof that a fact is not available to the intruder, we need to show that this specific fact has a characterizing property that enables its generation, and the intruder does not have that property. To achieve this, we assign values or *rank* to each fact, these ranks will depend on the protocol itself, the initial knowledge and capabilities of the intruder, and finally the facts that we want to reveal to the intruder [75].

We use the above definition and introduce the notion of a **sound** rank function, which is defined as follows:

Definition 3.3.2 *a rank function is* ***sound*** *if it satisfies the following condition: applying an event on the intruder's initial set of knowledge will not generate a zero rank:*

$\forall m \in \mathbb{K}_0, e \ \in \mathbb{E}, \rho(m) > 0$ *and* $\rho(e(m)) > 0$*, where* e *is an event in the set of protocol events.*

This intuitively means that the rank function will not generate or add any inconsistency to the group model. Therefore, it will guarantee that zero ranks can only be generated as a result of problems in the protocol.

It is necessary to verify that protocol participants cannot generate non-positive ranks. The appropriate rank function we choose to apply on the protocol should be sound. We define a set of rank functions with a number of requirements, which will be used to prove the correctness of the rank function.

Definition 3.3.3 *a rank function is* ***initially sound*** *if it satisfies these three requirements:*

1. $\forall m \in \mathbb{M}, \ \rho(m) >= 0$, there are no negative ranks generated by the system.

2. $\forall m \in \mathbb{K}_0, \ \rho(m) > 0$, intruder initial knowledge must be of positive rank.

3. $\forall m \in \mathbb{S}, \ \rho(m) = 0$, all secret messages must have a zero rank.

Definition 3.3.4 *Two events are **invertible** if each one is the inverse of the other.*

$e_2(e_1(m_1)) = m_1$, *where* $e_1, e_2 \in \mathbb{E}$, *and* $m_1 \in \mathbb{M}$.

Definition 3.3.5 *a rank function is **invertible** for all invertible events of inference rules.* $e_2(e_1(m_1)) = m_1 \Rightarrow \ \rho(e_2(e_1(m_1))) = \rho(m_1)$, *where* $e_1, e_2 \in \mathbb{E}$, *and* $m_1 \in \mathbb{M}$, *and any user of the system cannot apply an invertible event unless he/she is able to apply the inverse.*

Definition 3.3.6 *a rank function is **bounded** if* $\rho(m) - 1 \leq \rho(e(m)) \leq \rho(m) + 1$, *where* $e \in \mathbb{E}$, *and* $m \in \mathbb{M}$.

Theorem 3.3.1 *Rank Function Soundness*

A rank function is sound, if it is initially sound, invertible and bounded.

The theorem states that a rank function with the above specifications is consistent. It ensures that the zero rank cannot be generated by the initial knowledge of the intruder, or by the definition of the rank function for the events. In other words, applying each single event separately on the set of intruders initial knowledge will not generate a zero rank, simply because a secret is not revealed to the intruder. Formally, $\forall m \in \mathbb{K}_0, \ \rho(m) > 0$ and $\forall m \in \mathbb{K}_0, e \in \mathbb{E}, \ \rho(e(m)) > 0$. We use $\mathbb{R}$ to represent the set of all sound rank functions.

Proof. We prove this theorem using absurdum, by assuming that the rank function evaluates to zero then we show that for all possible execution events, there exists no message in the intruder's initial set of knowledge that can generate this zero rank.

Assume there exists $m \in \mathbb{M}$ such that $\rho(m) = 0$, the message m is either in the intruders initial set of knowledge (case (a) below) or generated after applying on single event on a message in the intruder's initial set of knowledge (case (b) below), therefore, we can write:

$$\rho(m) = 0 \ \Rightarrow \exists \, m' \in \mathbb{K}_0\text{:} \quad \begin{cases} (a) : \ m = m' \ \text{ or} \\ (b) : \ m = e(m') \end{cases}$$

Now we consider both cases and show the contradiction of the assumption:

For case (a): $m = m'$, since the rank function ρ is *initially sound* by definition, then there is a clear contradiction which can be stated as follows:

Only messages in $\mathbb{S}$ have the rank zero: $\rho(m') = 0 \ \Rightarrow m' \in \mathbb{S}$. However, messages in $\mathbb{S}$ are not in $\mathbb{K}_0$: $m' \in \mathbb{S} \Rightarrow m' \notin \mathbb{K}_0$. Which contradicts the assumption stated above: $m' \in \mathbb{K}_0$. Therefore $\rho(m') > 0$ is valid.

For case (b): $m = e(m')$, and $\rho(m) = 0$: the message m is generated after the application of one single event e on a message m' from $\mathbb{K}_0$.

Since ρ is bounded, then we can say that the rank of the message m is bounded by the rank of the message m', we can write this as follows:

$m = e(m') \Rightarrow \rho(m') - 1 \leq \rho(m) \leq \rho(m') + 1$

By assumption, we have $\rho(m) = 0$. This means that either $\rho(m') = 0$ or $\rho(m') = 1$ (from above inequality). Now we consider both cases: The case where $\rho(m') = 0$ and $\rho(m') = 1$. The first case is similar to case (a) above, and will lead to the same contradiction.

We consider the second possibility: $\rho(m') = 1$, $m = e(m')$ and $\rho(m') = \rho(m) + 1$.

ρ is *invertible*, therefore, there is an event e' that we can apply on the message m to generate the message m', we can write:

$m = e(m')$, $m' = e'(m)$ and therefore $m' = e'(e(m'))$ (which is a typical invertible relation in encryption).

We have $\rho(m') = 1$ and $m' = e'(m)$ therefore $\rho(m) = \rho(m') - 1$

which means that the message m' is generated after one single application of an event e' in the set of events on the message m: $m' = e'(m)$.

If the intruder can apply one single invertible event, then he/she can apply the other one. Since the events e and e' are invertible, and the intruder can apply e on m' to generate m, therefore he/she can also apply the event e' on m to generate m'.

This means that $e, e' \in \mathbb{E}$, $m' \in \mathbb{K}_0$, and $m \in \mathbb{S}$, so $\rho(m) = 0 \Rightarrow m \notin \mathbb{K}_0$ or $e \notin \mathbb{E}$ which contradicts the assumption stated above.

The fact that the intruder cannot generate secret knowledge from its initial knowledge (without executing the protocol), i.e., the intruder cannot decrypt a message encrypted with a secret key.

□

Chapter 4

Rank Theorems

4.1 Introduction

Even though group protocols intuitively claim immunity against forward and backward secrecy, the formal analysis and verification of these properties have received little attention in the research community. Theorem proving, among other formal methods, can be used in this context because of the ability of the logics behind it to capture and model special features of group key protocols. In order to use theorem proving, the protocol specifications and the security property should be modeled in the logic of the theorem prover, however this is not a straightforward task.

Theorem proving has been used for the verification of security protocols. The inductive approach was introduced by Paulson [68] and used by Bella *et al.* [11] to prove properties for the Kerberos authentication system, then, for the verification of Secure Electronic Transaction (SET) protocol [10]. Later on, Bella [9] applied the inductive approach to the verification of security protocols that make use of smart cards. These approaches could successfully handle complex parts of these protocols, however, they are still unable to handle forward and backward secrecy.

In this chapter, a rank theorems based method is presented to reason about group key protocols and their secrecy properties. Rank functions 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 as depicted in Figure 4.1. 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 the 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 then 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.

In order to prove the correctness of a specific property, we need to prove that its corresponding rank theorem is correct with respect to the protocol model. This is necessary to prove that a rank theorem implies the correctness of the security property it models. The final step is to mechanize the proof through PVS theorem prover through the implementation of events of the protocol in PVS, and then use PVS theorem proving strategies to construct the correctness of the rank theorem.

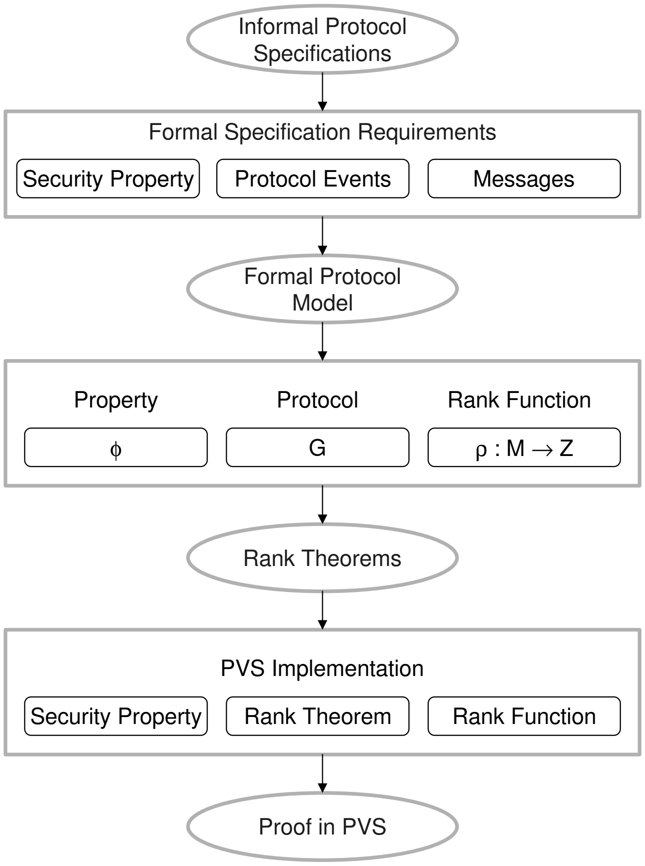


Figure 4.1: Overview of the Rank Theorems based Methodology

The rank theorem is composed of a security property ϕ, a rank function ρ, and a group protocol mode G, as illustrated in Figure 4.1. This is a general rank theorem that is used to instantiate specific ones for forward secrecy, backward secrecy, and then key independence under the condition that only two members can collude.

4.2 Rank Theorems Secrecy Properties

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. This means that these facts cannot be leaked to dishonest users.

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 $m2 = \mathit{encrypt}(m1, \mathit{key})$ then $\rho(m2) = \rho(m1) + 1$, where $m1, m2$, and $\mathit{key} \in \mathbb{M}$. Similarly, we define ρ for the *decrypt* event as follows: if $m2 = \mathit{decrypt}(m1, \mathit{key})$ then $\rho(m2) = \rho(m1) - 1$.

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:

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

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 prove 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:

4.2.1 Rank Theorem

Theorem 4.2.1 *Rank Theorem.*

$\forall m \in \mathbb{K}, \rho(m) > 0 \Rightarrow \mathbb{G}_t \models \phi$*, where* $m = \tau(\mathbb{E}, \mathbb{M})$*,* $\tau \in \mathbb{T}$*, and* ρ *is a sound rank function.*

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 $m \notin \mathbb{K}_0 \Rightarrow \exists \tau \in \mathbb{T} : m = \tau(\mathbb{E}, \mathbb{M})$, therefore,

$\tau(\mathbb{E}, \mathbb{M}) \leadsto m$ is valid. Then we can write

$\exists \tau \in \mathbb{T} : \tau(\mathbb{E}, \mathbb{M}) \leadsto m \Rightarrow \rho(m) = 0$, and so

$\exists \tau \in \mathbb{T} : \tau(\mathbb{E}, \mathbb{M}) \leadsto m \Rightarrow m \in \mathbb{S}$, this means

$\rho(m) = 0 \Rightarrow \neg\phi$, and so

$\rho(m) = 0 \Rightarrow \mathbb{G}_t \not\models \phi$

□

This general theorem is used in order to define a rank theorem for the forward secrecy, backward secrecy and key independence properties.

4.2.2 Rank Theorem for Forward Secrecy

Forward secrecy property, ϕ_f, is defined based on the formal specifications model, presented previously, as follows:

$$\phi_f = \forall m \in \mathbb{S}, I \in \mathbb{G}_t \; \Rightarrow \neg\exists\tau \in \mathbb{T} : \tau(\mathbb{E}, \mathbb{M}) \rightsquigarrow m$$

The rank function ρ_ϕ that maps the set of all messages to the appropriate ranks is defined as follows, where $\forall i \in \mathbb{Z} : t + i >= 0$ and $t - i >= 0$.

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

This means that for the validity of forward secrecy, we give rank zero to all the messages in the set of secret messages $\mathbb{S}$, such as secrets shared between users and servers, and all group 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/her initial set of knowledge.

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

Theorem *Rank Theorem for Forward Secrecy Property*

$\forall m \in \mathbb{K},\; \rho_\phi(m) > 0 \Rightarrow \mathbb{G}_t \models \phi_f$, *where* $m = \tau(\mathbb{E}, \mathbb{M})$ *and* $\tau \in \mathbb{T}$.

This theorem is an instantiation of the main rank theorem defined above under the same conditions, and its proof is the same.

4.2.3 Rank Theorem for Backward Secrecy

For backward secrecy, we define a rank function as follows:

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

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

Theorem *Rank Theorem for Backward Secrecy Property*

$\forall m \in \mathbb{K},\ \rho_b(m) > 0 \Rightarrow \mathbb{G}_t \models \phi_b$, *where* $m = \tau(\mathbb{E}, \mathbb{M})$ *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. In addition, two theorems, once are proven to be correct, can be used to prove collusion property. For this purpose we define a rank theorem for collusion property based on the above two theorems as follows:

4.2.4 Rank Theorem for Key Independence

In order to define a theorem for key independence, and be able to prove it efficiently, we aim at applying a simple and valid assumption on this property. We assume that two members are colluding their knowledge, therefore, the subset of known keys can be divided into two subsets: $\mathbb{K}_f$ and $\mathbb{K}_b$. This simplification leads to proving key independence on top of forward and backward secrecy, i.e., if the protocol satisfies both forward and backward secrecy simultaneously.

This assumption excludes specific cases, where more than two members may collude their knowledge. However, key independence is concerned with the case where two group users may cooperate to achieve access to secret shares. Moreover, the case of combining the intruders' knowledge in forward and backward secrecy can be generalized for more than two members. Based on this assumption, we define a rank theorem for key independence as follows:

Theorem *Rank Theorem for Key Independence Property*

$\mathbb{G}_t \models \phi_c \, iff \; \mathbb{G}_t \models \phi_f \wedge \mathbb{G}_t \models \phi_b$

Passive adversaries who know a proper subset of group keys cannot discover any other group key.

Proof. For key independence property, ϕ_c, the subset of group keys known by two adversaries can be included in two disjoin sets. Each one of these two subsets is equivalent to the subsets of keys in case of forward and backward secrecy. Therefore, if the group protocol model is correct for the adversary's knowledge sets in both forward and backward secrecy, it is correct for the adversaries' knowledge set in key independence.

□

This theorem is dependent on forward and backward secrecy, and it can be proven in the implementation once these latter are proven to be correct.

4.3 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.

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, which 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.

Then we define the prototypes for the events protocols can execute, which includes the normal events, like send, receive, encrypt, and decrypt, in addition to the dynamic events

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

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:

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

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

In order to update ranks of newly generated messages from events, we need lemmas

```
rank (msg): RECURSIVE int =

CASES msg OF

   userid(msg)   : 1,

   nonce (msg)   : IF msg = Ni THEN 1 ELSE 0 ENDIF ,

   public (msg) : 1,

   secret (msg) :  0 ,

   key(msg)      : IF secretKey(m) THEN 1 ELSE 0 ENDIF,

   inInitKnldg (msg) : 1

ENDCASES
```

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.

```
updateRank(event,m1,m2, key, u1,u2) : nat =
   CASES event OF
      conc(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
```

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

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

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

The last lemma states that if applying two events 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
```

The rank function for backward secrecy property initializes every message in the intruders initial set of knowledge and all the messages in the set of secret messages in the similar fashion we defined the rank function for forward secrecy. Here we assign a positive rank for old keys that we assume were generated when the intruder was part of the group.

```
rankBack (msg): RECURSIVE int =

CASES msg OF
   userid(msg) : 1,
   nonce (msg) : IF m = Ni THEN 1 ELSE 0 ENDIF ,
   public (msg) : 1,
   secret (msg) :  0 ,
   key(msg) : IF oldKey(msg) THEN 1 ELSE 0 ENDIF,
   inInitKnldg (msg) : 1
ENDCASES
```

4.4 Application: Enclaves Protocol

In this section, we apply the proposed rank theorems method on the Enclaves group key protocol. First, we provide the description of the protocol, then the embedding and verification in PVS.

4.4.1 Enclaves Protocol Description

Enclaves [33] 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. We intend to use the PVS model of the Byzantine agreement introduced in

[53], and present a single model in PVS that includes the Intruder, group key management, and security properties, in addition to the Byzantine agreement.

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.

The group-key management protocol in Enclaves relies on secure secret sharing. Each of the n leaders knows only a share of the group key, and at least $f + 1$ shares are required to reconstruct the key. Any set of no more than f shares is insufficient. This ensures that compromise of at most f leaders does not reveal the group key to the attacker.

In Enclaves, a new secret s and new shares must be generated whenever the group changes. This must be done online and without a dealer, to avoid a single point of failure. The n shareholders can individually compute their share of a common secret s without knowing or learning s. One can compute s from any set of $f + 1$ or more such shares, but f shares or fewer are not sufficient. The shareholders are the group leaders and $\tilde{g}$ is derived from the group view using a one-way hash function. Leader L_i computes its share s_i using a share-generation function S, the value $\tilde{g}$, and a secret x_i that only L_i knows: $s_i = S(\tilde{g}, x_i)$. Leader L_i also gives a proof that s_i is a valid share for $\tilde{g}$. This proof does not reveal information about x_i but enables group members to check that s_i is valid [33].

The secrecy properties of the protocol rely on the hardness of computing discrete logarithms in a group of large prime order which is constructed by selecting two large prime numbers p and q such that $p = 2q + 1$. The numbers $x_1, \ldots, , x_n$ are distributed secretly to the n leaders $L_1, \ldots, L_n$, respectively. A generator g and the elements $g_1, \ldots, g_n$ are made public. They must be known by all users and leaders. Leaders send hashed values of a combination of prime number generator and the secret share to all users, who can compute and verify the secret shares, and then use it to compute the key from the set of received messages from leaders.

In the current PVS model, we made some assumptions about the protocol, including abstracting the hash functions and the mathematical (exponentiation and module operations) computations in the 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 threat model, we assume intruders can monitor the network and choose to send messages on the network, either randomly or at their choice, in addition, they can block, modify, or insert messages. Finally, we assume 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 PVS implementation was applied on the forward secrecy property, backward secrecy property, and then on top of these two theorems, we define a theorem for collusion property.

4.4.2 PVS Implementation

We formalized the protocol events in PVS, utilizing previous implementations by [52] and [33], including all the operations that can be executed on the group. On top of these implementations we define and verify theorems for forward, backward secrecy and collusion property. 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
  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, the execution steps of the protocol taken by a user and leaders in order to complete the protocol are shown below [33].

```
  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, the reachable states in the protocol are defined as lemmas, 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, a preposition is defined 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 [33].

```
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.

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

```
forward_secrecy : THEORY
  BEGIN
      intKnldg  : VAR set[MESSAGE]
      msg       : VAR MESSAGE
      T         : Event

    fwd_secrecy_property: LEMMA

     forward_secrecy(intKnldg,msg,T) =
         FORALL msg,T : inSet(intKnldg,msg) and
                    Reachable(step_00, T)   and
                    Protocol?(msg,intKnldg)

                        IMPLIES rank(msg) > 0

  END forward_secrecy
```

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

Similarly, we can instantiate a rank theorem for backward secrecy and check its validity in the protocol states. However, here the rank function for backward secrecy is used. The lemma should guarantees that any message in the intruder knowledge will remain positive.

Finally, we provide the definition of collusion property based on the definition of forward and backward secrecy.

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 : THEORY
  BEGIN
      intKnldg  : VAR set[MESSAGE]
      msg       : VAR MESSAGE
      T         : Event

     bwd_secrecy_property: LEMMA

     backward_secrecy(intKnldg,msg,T) =
         FORALL msg,T : inSet(intKnldg,msg) and
                    Reachable(step_00, T) and
                    Protocol?(msg,intKnldg)

                          IMPLIES rankBack(msg) > 0

  END backward_secrecy
```

backward secrecy property efficiently 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.

4.5 Summary

In group key protocols, properties like forward and backward secrecy are very important for protocols correctness, however, they did not receive enough attention in the literature. This chapter presents rank theorems for forward and backward secrecy verification based on the notion of rank functions. The correctness of the rank theorems is established by first establishing the soundness of rank functions, then by showing that the rank theorem implies the correctness of the security property with respect to the group protocol model.

```
collusion_secrecy : THEORY
  BEGIN
      intKnldg  : VAR set[MESSAGE]
      msg       : VAR MESSAGE
      T         : Event

        collusion_property: LEMMA

        collusion_secrecy(intKnldg,msg,T) =
          FORALL msg,T : inSet(intKnldg,msg) and
                    Reachable(step_00, T)   and
                    Protocol?(msg,intKnldg)

                 forward_secrecy(intKnldg,msg,T) and

                 backward_secrecy(intKnldg,msg,T)

  END collusion_secrecy
```

Rank theorems are implemented in PVS to enable and mechanize the verification procedure of the Enclaves protocols. Forward and backward secrecy rank theorems are verified, then on top of these two, a rank theorem for key independence is established under certain assumptions.

The implementation of the rank theorems methods, from our experiences, requires a considerable amount of effort and time in theorem proving. To enhance the performance of this technique we combine the rank function into the protocol events. This resulted in a rank functions based inference system, an improved method to reason about messages and events at the same level of higher-order logics. The rank functions based inference system is presented in the next chapter.

Chapter 5

Rank Functions based Inference System

5.1 Introduction

In theorem proving, trying to prove the correctness of an incorrect theorem may not lead to a result, i.e., an indication that the theorem is incorrect. Therefore, it is necessary to be able to reason about both existence and absence of attacks. Since rank theorems are based on higher order logics, which is incomplete, the rank theorems are appropriate for reasoning about the correctness of the protocol, i.e, absence of attacks. However, reasoning about existence of attacks is not practical since the rank theorem implies that the rank function must always be non-zero. In addition, implementing rank theorems required a lot of user interaction with the verification tool, mainly because of the separation of rank functions from protocol events.

To overcome these limitations, this chapter combines protocol events with rank functions in order to obtain a rank functions based inference system. The inference system is composed of a set of inference rules over rank functions, where, every rule can be applied in order to generate new knowledge and assign new ranks to these generated messages. Figure 5.1 shows the steps of the verification methodology, where the inference system is composed of a set of inference rules, each combines protocol events and rank functions, the rules and security property are embedded in PVS. A special rule called *Attack* is defined to represent the bottom of the system, and, when executed, illustrates an attack in the protocol. This is illustrated on the verification of a generic Diffie-Hellman group protocol in the PVS.

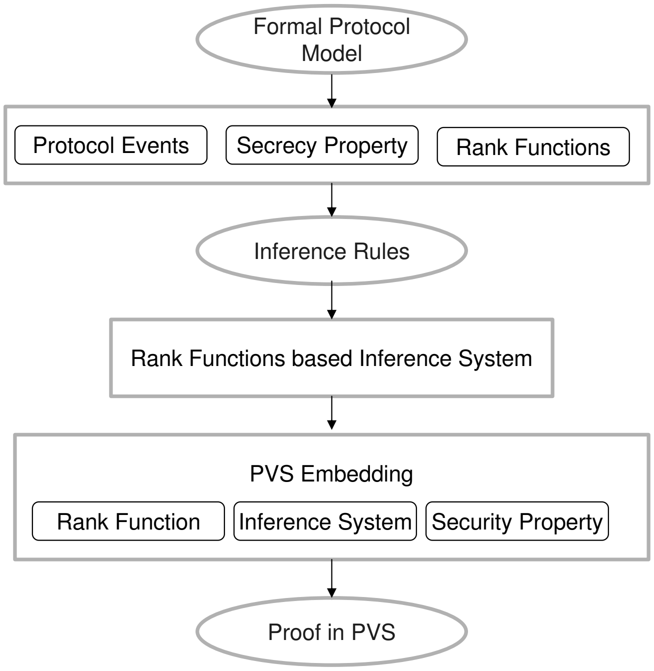


Figure 5.1: Overview of the Inference System based Methodology

Many efforts in the literature addressed DH style protocols. Kats and Shin [45] addressed the case of attacks by malicious insiders for authenticated key exchange protocols. Verma *et al.* [84] handled questions arising in cryptographic protocol verification, in particular in modeling group key agreement schemes based on Diffie-Hellman-like functions. Mazaré [57] proposed a symbolic model to analyze cryptographic protocols using bilinear pairing. Boreale and Buscemi [13] used another symbolic approach to verify protocols checking consistency of symbolic traces. Millen and Shmatikov [62] considered a symbolic approach to reason about GDH protocol style operators, such as exponentiation, with a bounded number of role instances.

These works address the algebraic properties of DH style protocols. However, these protocols can be subtle to non-algebraic attacks where the intruder takes advantage of messages interchange to establish the group key among members. Therefore, it is essential to prove the security of these protocol from both algebraic and non-algebraic points of view. In this chapter, we present a method to verify group protocols with respect to non-algebraic attacks.

5.2 Inference System

"Rules of inference are syntactical transformation rules which one can use to infer a conclusion from a premise to create an argument. A set of rules can be used to infer any valid conclusion if it is complete, while never inferring an invalid conclusion, if it is sound. A sound and complete set of rules need not include every rule in the following list, as many of the rules are redundant, and can be proven with the other rules" [36].

In this section, we present an inference system that consists of a set of inference rules. Every rule represents an event in the protocol. Rules have a precondition that has to be satisfied before they are applied. We define the pair $\langle m, \rho(m)\rangle$ to represent a message m and its rank $\rho(m)$. In the first rule, *Encryption*, if there is a message and its rank $\langle m, \rho(m)\rangle$, and a key and its rank $\langle k, \rho(k)\rangle$, the intruder can apply this rule and generate a new encrypted message. The rank of the new message is an increment of the rank of the old message if the key is secret, i.e., if $\rho(k) = 0$, then $\langle \{m\}_k, \rho(m) + 1\rangle$ is generated. On the other hand, if the key is not secret, the rank of the new message is equal to the rank of the old message, i.e, if $\rho(k) > 0$, then $\langle \{m\}_k, \rho(m)\rangle$ is generated. After executing this rule, the set of knowledge is updated with newly generated messages. The intruder should only be able to execute the second case, otherwise, if he/she knows $\langle k, 0\rangle$, then the rule $Attack$ below should be applied.

The rule *Decryption* works in opposite direction, given an encrypted message and its encryption key, $\langle \{m\}_k, \rho(\{m\}_k) \rangle$, and $\langle k, \rho(k) \rangle$, respectively. The system can apply this rule and generate a new message that has a rank that equals to the decrement of the original's message rank if the key is secret, i.e., if $\rho(k) = 0$, then $\langle m, \rho(\{m\}_k) - 1 \rangle$ is generated. This case should not be applied by the intruder, since he/she is not supposed to know $\langle k, 0 \rangle$, otherwise, the rule $Attack$ should be applied. If the key is not secret, the system generates a new message that has the same rank as the old one, i.e., if $\rho(k) > 0$, then $\langle m, \rho(\{m\}_k) \rangle$ is generated.

Similarly, we define *Compose*, where two message records are used to generate one new message with one rank. The *Decompose* rule is used to separate an already composed message. Another rule, *Expo*, is used to generate messages with the exponentiation operator. This operator is abstracted. Finally, the rule *Attack* is executed when there is a message of rank zero in the intruders set of knowledge. *Attack* is defined with a precondition, such that, when executed by the intruder, it indicates the occurrence of an attack by reaching the bottom of the system $\perp$.

The intruder, by executing these rules on the set of knowledge $\mathbb{K}$, generates new knowledge with new ranks and updates his/her set. For this system to work, we assume the fairness of executing these rules, i.e., the intruder will not keep using the rule *compose* forever, but other rules will have their chance to be executed, specially the rule *Attack*. The set of rules in the inference system are:

For this inference system, we use the event $Encr(m, k)$ to represent a message m that is encrypted w.r.t. a symmetric encryption algorithm with the key k. The event $Decr(Encr(m, k), k)$ represents decrypting a message that is already encrypted, where the same key used for the encryption is to be used for the decryption event. These two events are $invertible$, therefore $m = Decr(Encr(m, k), k)$. The event $Comp(m1, m2)$ represents

Rule1: *Encryption:* $\frac{\{\langle m,\rho(m)\rangle,\langle k,\rho(k)\rangle\}}{\{\langle m,\rho(m)\rangle,\langle k,\rho(k)\rangle\}\cup\{\langle\{m\}_k,\rho(m)+1\rangle\}}\ \ \rho(k)=0$

$\frac{\{\langle m,\rho(m)\rangle,\langle k,\rho(k)\rangle\}}{\{\langle m,\rho(m)\rangle,\langle k,\rho(k)\rangle\}\cup\{\langle\{m\}_k,\rho(m)\rangle\}}\ \ \rho(k)>0$

Rule2: *Decryption:* $\frac{\{\langle\{m\}_k,\rho(\{m\}_k)\rangle,\langle k,\rho(k)\rangle\}}{\{\langle\{m\}_k,\rho(\{m\}_k)\rangle,\langle k,\rho(k)\rangle\}\cup\{\langle m,\rho(\{m\}_k)-1\rangle\}}\ \ \rho(k)=0$

$\frac{\{\langle\{m\}_k,\rho(\{m\}_k)\rangle,\langle k,\rho(k)\rangle\}}{\{\langle\{m\}_k,\rho(\{m\}_k)\rangle,\langle k,\rho(k)\rangle\}\cup\{\langle m,\rho(\{m\}_k)\rangle\}}\ \ \rho(k)>0$

Rule3: *Compose:*
$\frac{\{\langle m_1,\rho_1\rangle,\langle m_2,\rho_2\rangle\}}{\{\langle m_1,\rho_1\rangle,\langle m_2,\rho_2\rangle\}\cup\{\langle Comp(m_1,m_2),min(\rho_1,\rho_2)\rangle\}}$

Rule4: *Decompose:*
$\frac{\{\langle Comp(m_1,m_2),\rho_1\rangle\}}{\{\langle Comp(m_1,m_2),\rho_1\rangle\}\cup\{\langle m_1,\rho_1\rangle,\langle m_2,\rho_1\rangle\}}$

Rule5: *Expo:* $\frac{\{\langle m_1,\rho_1\rangle,\langle m_2,\rho_2\rangle\}}{\{\langle m_1,\rho_1\rangle,\langle m_2,\rho_2\rangle\}\cup\{\langle {m_1}^{m_2},\rho_1-1\rangle\}}$

Rule6: *Attack:* $\frac{\{\langle m,0\rangle\}}{\perp}\ \ m\in\mathbb{K}$

two composed messages by concatenation. The function min gives the minimum rank from two given ranks.

5.2.1 Soundness and Completeness

In this subsection we define theorems for the soundness and completeness of the approach. The first theorem states that if we can find a message in the set of knowledge of the intruder that has the rank zero, or equivalently, if the rule *attack* of the inference system is applied, then there is an *attack* in the system. We assume *fairness* in applying the inference rules.

Theorem 5.2.1 *Soundness*

Let $\mathbb{G}$ be a security protocol, let ρ be a sound rank function, and let $\mathbb{K}_0$ be the set of the initial knowledge of the intruder. Then, the protocol $\mathbb{G}$ has an attack if the inference rule $attack$ can be applied in a fair inference system.

$\exists m\in\mathbb{K}:\rho(m)=0$ *and* $\rho\in\mathbb{R}\Rightarrow\exists attack$ *in* $\mathbb{G}$.

Proof. We prove this theorem by deduction, where we assume the left hand side and deduce the right hand side of the theorem.

Assume there exists $m \in \mathbb{K}$ such that $\rho(m) = 0$

Given that $m \in \mathbb{K}$, then we have $m = m_0 \in \mathbb{K}_0$, $m = e(m_0)$, or $m = T_n(\mathbb{E}, \mathbb{M})$. However, $\rho(m) = 0 \Rightarrow m \notin \mathbb{K}_0$, the rank function is sound since $\rho \in \mathbb{R}$.

$\rho(m) = 0 \Rightarrow \nexists\,(e \in \mathbb{E}, m_0 \in \mathbb{K}_0) : m = e(m_0)$, since the rank function is sound.

It is clear now that there exists $m \in \mathbb{K}$ such that $\rho(m) = 0 \Rightarrow \exists T_n \in \mathbb{T}$ such that $m = T_n(\mathbb{E}, \mathbb{M})$, which means that there exists a trace the intruder can execute to compute m.

Also $\rho(m) = 0 \Rightarrow m \in \mathbb{S}$, since only messages in $\mathbb{S}$ have the rank zero. Hence, we find that $m \in \mathbb{S}$ and $m \in \mathbb{K}$. Therefore, the rule $attack$ can be applied. Then, we can write: $attack \Rightarrow m \in \mathbb{S}$ and $m \in \mathbb{K}$. This means that there exists an attack in the system.

□

The following corollary is deduced from the above theorem and states that if there is no attack in the system, then a sound rank function will be greater than zero. It can be viewed as the complementary case of the above theorem.

Corollary 5.2.1 *Absence of Attack*

Assuming the same conditions as Theorem 5.2.1, if the protocol $\mathbb{G}$ has no attack, then the rule $attack$ will never be applied in a fair inference system.

$\nexists attack$ *in* $\mathbb{G} \Rightarrow \forall m \in \mathbb{K} : \rho(m) > 0$.

The second theorem states that if there is no message in the set of knowledge of the intruder that has the rank zero, or equivalently, if the rule *attack* of the inference system can never be applied, assuming *fairness* of the strategy application of inference rules, then there is no *attack* in the system.

Theorem 5.2.2 *Completeness*

Assuming the same conditions as Theorem 5.2.1, if the rule $attack$ *cannot be applied in a fair inference system, then the protocol* $\mathbb{G}$ *has no attack.*

$\forall m \in \mathbb{K} : \rho(m) > 0,\ \rho \in \mathbb{R} \Rightarrow \nexists attack$ *in* $\mathbb{G}$.

Proof. We prove this theorem by absurdum. We assume the right hand side is false and deduce a contradiction to the left hand side of the theorem.

Assume there exists an $attack$ in the system, then we can write:

$\exists attack \Rightarrow \exists m$ such that $m \in \mathbb{K}$ and $m \in \mathbb{S}$.

A message m in the intruder's set of knowledge $\mathbb{K}$ means that either the message is in his/her initial set of knowledge $\mathbb{K}_0$, generated by applying one single event in $\mathbb{E}$, or is generated after applying a trace in $\mathbb{T}$.

$m \in \mathbb{K} \Rightarrow m = m_0 \in \mathbb{K}_0$, $m \in \mathbb{K}_1 = \mathbb{E}(\mathbb{K}_0)$, or $m = T_n(\mathbb{E}, \mathbb{M})$.

However, since $\rho \in \mathbb{R}$ is sound and $m \in \mathbb{S}$ then $m \notin \mathbb{K}_0$ and $m \notin \mathbb{K}_1$.

Therefore, $m = T_n(\mathbb{E}, \mathbb{M})$

Since $m \in \mathbb{S}$, then the rank of this message $\rho(m) = 0$.

So $\exists attack \Rightarrow \rho(m) = 0$.

Therefore, we conclude that $\rho(m) > 0 \Rightarrow \nexists attack$.

□

Corollary 5.2.2 *Detecting Attacks*

Assuming the same conditions as Theorem 5.2.1, if the rule $attack$ *can be applied in a fair inference system, then the protocol* $\mathbb{G}$ *has an attack.*

$\exists m \in \mathbb{K} : \rho(m) = 0$ *and* $\rho \in \mathbb{R} \Rightarrow \exists attack$ *in* $\mathbb{G}$.

This corollary states that when the rank function evaluates to zero, then there exists an attack in the protocol. Theorems 5.2.1 and 5.2.2 and Corollaries 5.2.1 and 5.2.2 provide the formal link between the protocol model and the inference system.

The inference system can prove that an attack exists in the protocol using Theorem 5.2.1. However, the limitation of the Soundness Theorem comes in the type of implementation that will be used. In case of theorem proving, there is no guarantee that the attack can be generated. This is a general problem and is applicable on any approach for tool supported verification this type of protocols.

When reasoning about absence of attacks in the protocol, the inference system can diverge in case we apply the Theorem 5.2.2. The inference system may terminate with a result, depending on the nature of the protocol and the strategies used while conducting theorem proving. However, there is no guarantee for termination because of two reasons: first, the type of problem we are trying to solve has unbounded number of participants, and unbounded message space. Second, the lower level implementation method, theorem proving, does not guarantee termination, which means the inference system may run infinitely without reaching a result.

We still can reason about absence of attacks in protocols using Theorem 5.2.2. An *indirect* proof can be generated by proving that the strategy is fair and the application of our inference system diverges. This *indirect* proof can be achieved by generating partial proofs that affirm that the inference system will diverge when the applied strategy is fair. We believe that in order to be apply this approach, we have to provide an implementation for the inference system that allows partial proofs based on divergence. This later issue will be considered for further study.

This method complements the rank theorems method since it is more efficient in reasoning about the existence of an attack, a limitation in the rank theorems method.

5.3 PVS Embedding of the Inference System

In this section, we describe the embedding of the inference rules in PVS in order to be able to apply it on the target group protocol.

An important and challenging part is how to define the inference system, and how to instantiate it by the dishonest user. For this purpose, we represent each inference rule as a PVS deduction statement which allows the user who is executing these rules to compute new messages and add it to his/her own set of knowledge. In our case, the intruder is such user. These rules are defined based on the events of the GDH protocol. We abstract the mathematical power operation used in the protocol, since power operator is not supported in PVS. Therefore α^N is represented in PVS by the variable $alphaN$ and its rank function is defined of a type that maps $MESSAGE$ to int.

```
MESSAGE : TYPE ALPHA:  TYPE FROM MESSAGE nounc: int alpha: int

alphaN : ALPHA m: MESSAGE rankf: [MESSAGE -> int]
```

Next we define the appropriate inference rules for this protocol. These rules are used by the intruder in order to build his/her set of knowledge starting from his initial knowledge and applying one rule at a time. In addition to the rules *compose*, *encrypt* and *decrypt*, and *attack* described above we show the rule *expo* which is used to generate the α^N messages.

```
rule_compose(msg1: MESSAGE, msg2 : MESSAGE) : MESSAGE
   = (comp(msg1,msg2), min(rankf(msg1), rankf(msg2)))

rule_decomp(msg1: MESSAGE) : [MESSAGE, int, MESSAGE, int]
   = (left(msg1),rankf(msg1), (right(msg1), rankf(msg1))

rule_encr(msg1: MESSAGE, key: MESSAGE) : [MESSAGE, int]
   = (append(msg1,key), rankf(msg1)-1)
```

```
rule_decr(msg1: MESSAGE, key: MESSAGE) : [MESSAGE, int]
   = (extract(msg1,key), rankf(msg1)+1)

rule_expo(a:ALPHA,N:int) : [ALPHA, int]
   = (alphaN, rankf(a)-1)

rule_attack(msg: MESSAGE): bool =
   rankf(msg) = 0
```

The above inference rules are used in PVS interactively, where it is upon the user to choose which rule he/she can execute next. However, if the precondition is not satisfied for a specific rule, then it cannot be executed. This way, the user can ensure the fairness of the inference system. Decision procedures can be implemented to help in choosing which rule to apply next, however, this will depend on the protocol and security property under verification, otherwise, it will not be efficient in deducing the proof.

5.4 Application: GDH Protocol

In order to illustrate the proposed verification methodology, we consider the Group Diffie-Hellman (GDH) protocol [78, 32], which is a basic group key management protocol widely studied in the literature. In the first part, we show how to manually detect the attack in a step by step application of the inference system. Then we use the PVS theorem prover in order to implement the inference system and apply it on the protocol for two, three, and n-users. Throughout this work, we assume perfect cryptographic conditions, we analyze the key agreement nature of the protocol, and we abstract the algebraic exponentiation property of the protocol.

Although the protocol is not a challenging case study, since it has been studied quite enough in the literature, we use it as illustrative case to show the feasibility of our approach, not to prove something that has been already proved before. In addition, we provide, to the best of our knowledge, a new attempt to use theorem proving in the context of group key protocols verification. In the first part of this section we show how the inference system can be applied directly on the GDH protocol of four participants. We demonstrate how the attack is generated in the step by step application of the inference system. In the second part, we discuss the embedding and generation of the verification of the protocol using theorem proving in PVS.

5.4.1 Protocol Description and Attack Illustration

The protocol is used to generate and distribute a safe key between a group of members over a non-secure network. It consists of two stages: upflow and downflow. The first stage is used to collect contributions from all group members that will be used in calculating the group key. Given n members in the group: $P_1, P_2, ..., P_n$, who are willing to generate a secret key that will be used among them, the protocol works as follows: in the upflow stage, every intermediate member P_i receives a collection of intermediate values from member P_{i-1}, computes another value, by adding his/her own share of the key, appends it to the values he/she received, and forwards this information to the next group member P_{i+1}. In the downflow stage, the last member appends his own share of the key to every value he received and sends them back to previous members. This way, every member receives partial information to compute the key. The two stages of the protocol are

Stage 1 (Upflow): $M_i \longrightarrow M_{i+1}$

Round i: $\{\alpha^{\Pi(N_k|j\in[1,j])}|j \in [1,i]\}; i \in [1, n-1]$

Stage 2 (Downflow): $M_{n-i} \longrightarrow M_{n-i+1}$

Round $n-1+i$: $\{\alpha^{\Pi(N_k|j\notin[i,j])}|j \in [1,i]\}; i \in [1, n-1]$

For example, in a group of four members, the first member uses a generator α and a random number N_1, computes $\{\alpha^{N_1}\}$ and forwards it to member P_2. P_2 chooses a random number N_2 and computes $\alpha^{N_1N_2}$, then forwards $\{\alpha^{N_1}, \alpha^{N_1N_2}\}$ to P_3. P_3 computes $\alpha^{N_1N_2N_3}$ and forwards $\{\alpha^{N_1N_3}, \alpha^{N_1N_2}, \alpha^{N_1N_2N_3}\}$ to P_4. Now P_4 uses the last value and a random number he/she generates, N_4, to compute the group key $\alpha^{N_1N_2N_3N_4}$. For the downflow stage, the member raises all other values to N_4 and sends back to P_3 $\{\alpha^{N_4}, \alpha^{N_1N_2N_4}, \alpha^{N_1N_3N_4}\}$. P_3 uses the latest value $\alpha^{N_1N_2N_4}$ and his/her own random number N_3 to compute the key, raises the first value to N_3 and sends $\{\alpha^{N_3N_4}, \alpha^{N_1N_3N_4}\}$ back to P_2, who uses the last one to compute the key, then computes and sends $\alpha^{N_2N_3N_4}$ to P_1, who can compute the same key.

We choose a group of three members, P_1, P_2, and P_3, and apply our verification approach on the protocol, by defining the active intruder I, and executing the inference system by this intruder. The case of three members is similar to four or five members, however it is easier to illustrate and sorter to describe . The first step is to define the rank function ρ for the set of messages in the message space M as follows:

$$\rho(m) = \begin{cases} 0, & if\ m \in \{N_1, N_2, N_3, \alpha^{N_1N_2N_3}, \alpha^{N_iN_2N_3}, \\ & \qquad \alpha^{N_1N_iN_3}, \alpha^{N_1N_2N_i}, \alpha^{N_1N_2N_3N_i}\} \\ & \\ 1, & if\ m \in \{N_i, \alpha, \alpha^{N_1}, \alpha^{N_2}, \\ & \qquad \alpha^{N_3}, \alpha^{N_1N_2}, \alpha^{N_1N_3}, \alpha^{N_2N_3}\} \end{cases}$$

Here, $\alpha^{N_1N_2N_3}$ represents the group key members intend to generate, and N_i represent the intruders nounce. For the protocol to be correct, there should not be a way for an intruder to share a key with the rest of the members that can be considered as the group key, even if its different from the group key the members intend to generate, simply because the members have no clue about the final key, until each has enough shares from other members to compute it. Therefore, we consider $\alpha^{N_iN_2N_3}, \alpha^{N_1N_iN_3}, \alpha^{N_1N_2N_i}, \alpha^{N_1N_2N_3N_i}$ as assumed group keys that the intruder should not be able to share with the members making them believe it is the group key they intended to share when they started the protocol.

Then, we define the events that can be executes by members. This includes $send(m)$, $recv(m)$, $expo(m, n)$, $comp(m_1, m_2)$, $decomp(m_1, m_2)$, and $block(m)$. The latest is an event that can be executed only by the intruder. The rank function ρ can be defined for these events as follows:

$\rho(send(m)) \ = \ \rho(m)$

$\rho(recv(m)) \ = \ \rho(m)$

$\rho(expo(m, n)) \ = \ \rho(n) + 1$

$\rho(comp(m_1, m_2)) \ = \ \rho(min(m_1, m_2))$

$\rho(block(m)) \ = \ \rho(m)$

$\rho(m_1) = \rho(m_1.m_2)$, $\rho(m_2) \ = \ \rho(m_1.m_2)$ for $decomp(m_1.m_2)$ event.

The inference system is composed of an inference rule for each of these events in addition to the inference rule *Attack* defined above. The event $expo(m, n)$ models the exponent function used in the protocol. The intruder starts with an initial set of knowledge $\mathbb{K}_0 = \{N_i, \alpha\}$, and we assume he/she has the ability to fully monitor the network, send, receive, or block messages on his/her will. We also assume that signing messages is not used between members.

Even though the protocol is designed to work in the existence of a passive adversary, we illustrate the attack generation by applying the protocol in the existence of an *active* adversary I and three protocol members : $P1, P2$, and $P3$. This is because the inference system represents the model of the intruder. We divide the protocol interaction steps between members, including the adversary, into different phases according to the activity performed by the adversary for easier understanding of the attack generation.

The upflow stage of the protocol is started in the first phase, called $P1$ *Phase*, by member P_1, who uses α, N_1, and $expo(\alpha, N_1)$ to compute α^{N_1}, where $\rho(\alpha^{N_1}) = \rho(N_1) + 1 = 1$. P_1 sends this message to next user in the group, P_2. The intruder applies the inference rules corresponding to the event $recv(\alpha^{N_1})$, therefore, $\mathbb{K} = \mathbb{K}' \cup \{\alpha^{N_1}\}$. We use $\mathbb{K}'$ to represent the intruder's set of knowledge before he/she executes the inference rule. This is illustrated as:

I : $\mathbb{K}_0 = \{N_i, \alpha\}$

P_1: $expo(\alpha, N_i) \mapsto \alpha^{N_i}; \mathbb{K} = \{N_i, \alpha, \alpha^{N_i}\}$

P_1: $expo(\alpha, N_1) \mapsto \alpha^{N_1}, \rho(\alpha^{N_1}) = \rho(N_1) + 1 = 1$

P_1: $send(\alpha^{N_1}); P_1 \rightarrow P_2$

The intruder similarly applies the inference rule $block(\alpha^{N_1})$, then, computes α^{N_i} and sends it to the member P_2, who computes $\alpha^{N_i N_2}$, uses the $compose$ rule to generate $\alpha^{N_i}.\alpha^{N_i N_2}$ and forwards it to the last member P_3. This is illustrated as

I : $recv(\alpha^{N_1}), block(\alpha^{N_1}); \mathbb{K} = \{N_i, \alpha, \alpha^{N_1}\}$

I : $send(\alpha^{N_i}); I \rightarrow P_2$

P_2: $recv(\alpha^{N_i});$

P_2: $expo(\alpha^{N_i}, N_2) \mapsto \alpha^{N_i N_2}$

P_2: $compose(\alpha^{N_i}, \alpha^{N_i N_2}) \mapsto \alpha^{N_i}.\alpha^{N_i N_2}$

P_2: $send(\alpha^{N_i}.\alpha^{N_i N_2}); P_2 \rightarrow P_3$

The intruder can receive this message and block it, then, composes and sends to P_3 the message $\alpha^{N_1}.\alpha^{N_i N_2}.P_3$ receives this latter message, decomposes it and uses the last term $\alpha^{N_i N_2}$ to compute the key $\alpha^{N_i N_2 N_3}$ believing it is the intended group key. At this point the intruder's set of knowledge is updated to the following: $\mathbb{K} = \{N_i,\ \alpha,\ \alpha^{N_1},\ \alpha^{N_1N_i},\ \alpha^{N_iN_2}\}$. This is the end of the upflow stage and illustrated as

$I\ :\ \mathit{recv}(\alpha^{N_i}.\alpha^{N_iN_2}), \mathit{block}(...); \mathbb{K} = \{N_i, \alpha, \alpha^{N_1}, \alpha^{N_i}.\alpha^{N_iN_2}\}$

$I\ :\ \mathit{decompose}(\alpha^{N_i}.\alpha^{N_iN_2}) \mapsto \alpha^{N_i}, \alpha^{N_iN_2}; \mathbb{K} = \mathbb{K}' \cup \{\alpha^{N_iN_2}\}$

$I\ :\ \mathit{compose}(\alpha^{N_1}, \alpha^{N_iN_2}) \mapsto \alpha^{N_1}.\alpha^{N_iN_2}$

$I\ :\ \mathit{send}(\alpha^{N_1}.\alpha^{N_iN_2}); I \to P_3$

Member P_3 then starts the downflow stage. He/she receives the message $\alpha^{N_1}.\alpha^{N_iN_2}$, and keeps $\alpha^{N_iN_2}$ to use it for group key calculation. Then, he/she computes, composes and sends the message $\alpha^{N_3}.\alpha^{N_1N_3}$ back to user P_2. The intruder will receive and block this message, and therefore, updates his/her knowledge such that $\mathbb{K} = \mathbb{K}' \cup \{\alpha^{N_3},\ \alpha^{N_1N_3}\}$. The intruder composes the message $\alpha^{N_3}.\alpha^{N_iN_3}$ and sends it to P_2. P_2 receives the message and he/she uses the last term $\alpha^{N_iN_3}$ to compute his/her key $\alpha^{N_iN_2N_3}$. Then he/she sends back to user P_1 the message $\alpha^{N_2N_3}$.

P_3: $\mathit{recv}(\alpha^{N_1}.\alpha^{N_iN_2})$

P_3: $\mathit{decompose}(\alpha^{N_1}.\alpha^{N_iN_2}) \mapsto \alpha^{N_1}, \alpha^{N_iN_2}$

P_3: $\mathit{expo}(\alpha^{N_iN_2}, N_3) \mapsto \alpha^{N_iN_2N3}$; P_3 generates a bad group key $\alpha^{N_iN_2N3}$

P_3: $\mathit{expo}(\alpha^{N_1}, N_3) \mapsto \alpha^{N_1N3}$;

P_3: $\mathit{expo}(\alpha, N_3) \mapsto \alpha^{N3}$;

P_3: $\mathit{compose}(\alpha^{N_3}, \alpha^{N_1N_3}) \mapsto \alpha^{N_3}.\alpha^{N_1N_3}$

P_3: $\mathit{send}(\alpha^{N_3}.\alpha^{N_1N_3})$; $P_3 \to P_2$

The intruder receives the message $(\alpha^{N_3}.\alpha^{N_1N_3})$, updates his/her set of knowledge, blocks the message. Then, he/she computes and sends $\alpha^{N_iN_3}$ to P_2 instead of the original message. P_2 in turn will use it to compute his/her key $\alpha^{N_2N_3N_i}$. The intruder updates his/her set of knowledge at this point, and it will be $\mathbb{K} = \mathbb{K}' \cup \{\alpha^{N_iN_3},\ \alpha^{N_iN_1N_3}\}$.

$I\ : recv(\alpha^{N_3}.\alpha^{N_1N_3}), block(...); \mathbb{K} = \mathbb{K}' \cup \{\alpha^{N_3}.\alpha^{N_1N_3}\}$

$I\ : decompose(\alpha^{N_3}.\alpha^{N_1N_3}) \mapsto \alpha^{N_3}, \alpha^{N_1N_3}; \mathbb{K} = \mathbb{K}' \cup \{\alpha^{N_3}, \alpha^{N_1N_3}\}$

$I\ : expo(\alpha^{N_3}, N_i) \mapsto \alpha^{N_iN_3};$

$I\ : compose(\alpha^{N_i}, \alpha^{N_iN_3}) \mapsto \alpha^{N_i}.\alpha^{N_iN_3}$

$I\ : send(\alpha^{N_i}.\alpha^{N_iN_3}); I \rightarrow P_2$

P_2: $recv(\alpha^{N_i}.\alpha^{N_iN_3})$

P_2: $decompose(\alpha^{N_i}.\alpha^{N_iN_3}) \mapsto \alpha^{N_i}, \alpha^{N_iN_3}$

P_2: $expo(\alpha^{N_iN_3}, N_2) \mapsto \alpha^{N_iN_2N3}$; P_2 generates a bad group key $\alpha^{N_iN_2N3}$

P_2: $expo(\alpha^{N_i}, N_2) \mapsto \alpha^{N_iN2}$

P_2: $send(\alpha^{N_iN_2}); P_2 \rightarrow P_1$

The intruder receives the message $\alpha^{N_2N_3}$, updates his/her set of knowledge, blocks the message, and instead, he/she sends $\alpha^{N_iN_3}$ to P_1 who, in turn will use it to compute his/her key $\alpha^{N_1N_3N_i}$. The intruder updates his/her set of knowledge at this point, and it will be $\mathbb{K} = \{N_i,\ \alpha,\ \alpha^{N_1},\ \alpha^{N_1N_i},\ \alpha^{N_iN_2},\ \alpha^{N_1N_3},\ \alpha^{N_iN_2N_3},\ \alpha^{N_iN_1N_3}\}$.

Now, given a fair system, the intruder can apply the inference rule *Attack*, since there is a message in his/her set of knowledge that has the rank zero.

Since this protocol provides no authentication, this attack can be generated in a more simple and trivial way, simply by allowing the intruder to play the rule of a group user, for instance, P_3. However, we choose to generate the attack using these complex steps to show

$I\ :recv(\alpha^{N_iN_2}), block(...)$

$I\ :expo(\alpha^{N_3}, N_i) \mapsto \alpha^{N_iN3};$

$I\ :send(\alpha^{N_iN_3}); I \rightarrow P_1$

$P_1: recv(\alpha^{N_iN_3})$

$P_1: expo(\alpha^{N_iN_3}, N_1) \mapsto \alpha^{N_iN_1N3}; P_1$ generates a bad group key $\alpha^{N_iN_1N3}$

$I\ :expo(\alpha^{N_1N_3}, N_i) \mapsto \alpha^{N_iN_1N3}; \mathbb{K} = \mathbb{K}' \cup \{\alpha^{N_iN_1N_3}\}$

$I\ :Attack(\langle \alpha^{N_iN_1N_3}, 0 \rangle) \mapsto \perp$

applicability of our inference system on such complex scenarios and its ability to generate such attacks when they exist.

5.4.2 PVS Implementation

Next, we define in PVS the sets of messages we used, including the set of all messages, secret messages, events, traces, intruders initial knowledge, intruders updated knowledge. We also define the dishonest user *I* and a set of *n* users who all together will participate in the protocol. We also define the intruder's initial set of knowledge to be α, N_i as stated above.

In order to conduct the verification for the secrecy property in PVS, we first considered the simple case of GDH protocol where two users are establishing the secret key. For this purpose, we define two users and a dishonest user and the set of messages used in the protocol. In addition, we show how the intruder updates his set of knowledge when a message is sent between two users and blocked by the intruder. For illustration purposes, we show parts of the protocol implementation including $send$ and $receive$ operations that take place between users. The *GDHP_update* function applies the inference rules on the set

```
    M: VAR set[MESSAGE]

    Key: VAR set[KEY]

    K: VAR set[MESSAGE]

    A: VAR USER

    I: VAR USER

    S: VAR set[MESSAGE]

    rule_initial(alpha: ALPHA)  = K_0 := [alpha,Ni]
```

of messages and then updates the intruders set of knowledge. For the functions *send* and *receive*, if the message is not blocked, then it is received at the destination user, the intruder add the message to his set of knowledge, and finally he/she updates this set by executing the function *update*. If the message is blocked, it is processed by the intruder.

```
GDHP_update?(K, I): bool =

    (FORALL m: K  do K := union(K, inf_rule(I,m)))

GDHP_send?(UserA, UserB, m) =

        if(!block(UserA,UserB,m) then GDHP_recv(UserA,UserB,m)

        addMsg(K,m)

        GDH_update(K,I)

GDHP_recv?(UserA, UserB, m) =

        addMsg(UserB.knldgSet,m)
```

The secrecy property we verify for this protocol is defined as a lemma stating that the protocol satisfies this secrecy property if it does not execute the inference rule attack. The property shows that when the protocol is initiated by a user and the intruder can execute the events of the protocol (rules in the inference system), then the intruder will be able to share a secret key between him/her and the user in the group. The property is defined as follows:

```
secrecy_prop_x : THEORY
  BEGIN
      secrecy_attack: LEMMA
         Reachable(rule_inital)AND knows(I,K_0)
                     IMPLIES Reachable(rule_attack)
  END forward_secrecy
```

In the next step we verify the same property by executing the inference system on a protocol between three users instead of two. The verification complexity and effort were more in this case, however there was no technical changes in the verification techniques and strategies used. In following we show how the property definition for the GDH protocol with three users in PVS:

```
secrecy_prop_3 : THEORY
  BEGIN
    A, B, C: VAR USER
    secrecy_attack: LEMMA
      Reachable(rule_inital)AND knows(I,K_0) AND GDHP(A,B,C)
                  IMPLIES Reachable(rule_attack)
  END secrecy_prop_3
```

The challenging part was to verify the N users case of GDH protocol. This was achieved by applying the same proof strategies used for the three users case for an array of n-users in order to show that the same attack can be generated in this case. In order to show that the intruder can generate the attack for this case, we use induction on the number of users, so we assume that the attack can be generated for n-users, then we show that the attack can be generated for $n+1$-users. This is straightforward in PVS, since the same inference rules that are applied in the n-users case can be iterated in the $n+1$-users case and then the additional user is treated in few more inference rules.

In the following, we show the secrecy property definition for the GDH protocol for an array for n users:

```
secrecy_prop_n : THEORY
  BEGIN
      n: VAR int
      users: array[n] of USER
      secrecy_attack: LEMMA
         Reachable(rule_inital)AND knows(I,K_0) AND GDHP(users)
                     IMPLIES Reachable(rule_attack)
  END secrecy_prop_n
```

Implementing and verifying this part in PVS required hundreds lines of code, including several proof strategies. We believe using the framework to verify similar protocols can be achieved in shorter time given the provided implementation of the inference system and the experience gained. As opposed to previous works, such is in [31], [57], [62], and [83], our approach gives a simple, natural and elegant proof strategy. We found that, under certain assumptions, the intruder can force members using the protocol to generate bad keys, which is a well known weakness point in the protocol. The results we achieved are very promising and we believe that our framework can be applied efficiently on protocols of similar complexity level.

It is true that Diffie-Hellman based protocols are considered as an algebraic protocols. Therefore, it could be argued that our approach is more appropriate for non-algebraic protocols, however, the non-algebraic framework described here is appropriate to model the distributive features of such protocols, while most other works analyze this class of protocols algebraically. We intended to keep our focus entirely within these non-algebraic features which has prompted us to make some necessary assumptions to be able to model and verify the protocol.

5.5 Summary

In this chapter, we combined protocol events with rank functions in order to obtain a rank functions based inference system. This method is distinguished from rank theorems in two aspects: first the inference system helps in reducing user interaction with the theorem prover, second, the inference system is designed to reason about the existence of attacks in the protocol, whereas, the rank theorems are more appropriate for reasoning about their absence.

Discovering if the protocol is vulnerable for attacks from an intruder is done by executing the inference system by a model of the intruder with specific assumptions about the protocol and the intruder. This method is applied on the example protocol in the existence of an active adversary. The case of a passive adversary is more restricted than an active one. In addition, it has been shown that a protocol that is secure in the passive setting can be considered secure in the active case [46].

The major shortcoming of this method, as well as the rank theorems, is its interactivity. In the next chapter, we intend to apply abstraction techniques on the protocol model in order to make automatic theorem proving feasible using first-order logic tools. This will reduce the complexity of the verification process and make it more automatic, but it will limit the applicability of the method on large scale and complex protocols.

Chapter 6

Event-B based Verification

6.1 Introduction

In the previous chapter, we introduced a rank functions based inference system for verification of secrecy in group key protocols that is implemented in higher-order logic theorem proving. Implementing the inference system, and similarly, the rank theorems, in higher order logic theorem proving, required a lot of effort and time, in addition, verifying properties is achieved interactively with the theorem proving tool because of the decidability problem on higher-order logic. In addition, because higher-order logic is incomplete and undecidable, there might be cases where the proof cannot be deduced. In order to provide some automation, first-order logic, which is complete and semi-decidable, should be used. In order to move from higher-order logics, to first-order logics, certain simplifications and assumptions should be applied.

Events-based verification of security protocols was used by Crazzolara and Winskel [26, 27] using mappings between process algebra, Petri nets, strand spaces and inductive models to prove an authentication property for security protocols. Butler [19] combined CSP and B method refinement in order to verify authentication property. Stouls and Potet

[80] proposed a method to automatically enforce an abstract security policy on a network. A different approach to achieve a similar objective was proposed in [12], where the authors addressed the proof-based development of system models satisfying a security policy. Andova *et al.* [6] presented a an events-based framework for easy verification of a large and useful class of security protocols built from smaller sub-protocols. They applied their framework to the composition of three protocols from security layer of WiMAX. However, these efforts did not address group key protocols, specifically their dynamics aspects and secrecy properties.

In order to model a group key protocol in event-B first order logic, the semantics of the event-B language should be formally related to the protocol model. We define well-formed conditions to guarantee that the event-B invariant is equivalent to the security property in the group key protocol model. These conditions are particular to the group key protocol model, and are essential to establish the equivalent event-B model. We show how an event-B model can be structured from group key protocols model and then used to give a formal semantics to protocols which support proofs of their correctness.

This method is illustrated on the tree based Group Diffie-Hellman protocol in order to verify group key secrecy and forward secrecy.

6.2 Event-B Semantics based Verification

It is a practical solution to verify a security property using model checking tools, when applicable. However, it is inconvenient because of two reasons: the state space explosion problem of model checking, and the limited expressiveness of proposition logic based-tools. Treating the problem at the first-order level requires applying a valid abstraction on the protocol in order to fit to the proving system. This abstraction should be based on a correct semantical link between the protocol model and the target model. In this method, we tackle this problem by using event-B as the target first-order model benefiting from the automation and the expressiveness of first order logic.

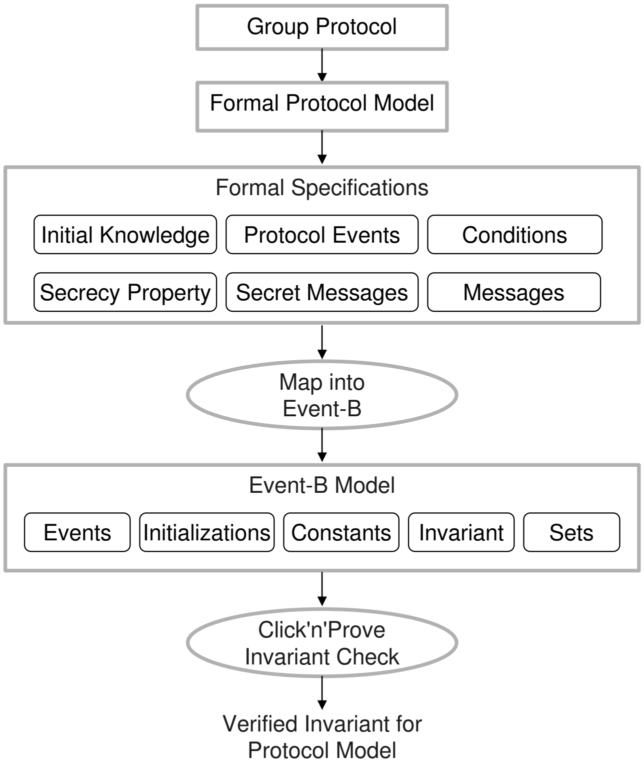


Figure 6.1: Overview of the Event-B Invariant based Method

The proposed verification methodology consists of a number of steps as shown in Figure 6.1. In the first step, the group key protocol is specified formally using the model presented in Chapter 3 in order to obtain precise protocol specifications. In addition, the secrecy property expected to be checked by the system is described informally. In the second step, the obtained specification is translated into event-B specification using mapping relations presented in Figure 6.2. From this mapping we obtain an event-B model that captures the features of the group protocol mode. Next, the secrecy property ϕ is specified as an invariant of the resulting event-B model I. Messages can be defined as a set with an enumeration of all possible secret and known messages. The intruder initial knowledge, $\mathbb{K}_0$, is directly defined as variable or set in the event-B *initialization* list. Secret messages are defined similarly. Protocol initial constraints, such as $\mathbb{K}_0 \subset \mathbb{M}$ and $\mathbb{S} \subset \mathbb{M}$, are defined as properties that will be included in the invariant. Protocol join or leave events are defined as event-B operations that update the intruder's knowledge and the set of secret messages, including the new generated key. Finally, the property is checked from the obtained global system specification using the event-B invariant checking tool Click'n'Prove.

In Figure 6.2, protocol events and execution traces are mapped into event-B events, messages generation conditions are mapped into events guards, and messages sets are used to generate event-B model constants properties. The initial knowledge is defined as event-B initializations, messages are mapped directly into sets, and finally the secrecy property is defined as an invariant for the event-B model. The generation of the target event-B model requires treating three parts: the static part which includes initializations and the constant properties of the protocol, the dynamic part that represents events of the protocol, and finally, enriching the resulting model with invariants describing the required secrecy properties.

6.3 Verification of Secrecy as Event-B Invariant

The event-B semantics is close to the protocol model semantics. This relationship is demonstrated by establishing a well-formed link between the semantics of both models. To achieve this link, we are interested in showing that if the invariant I holds for event-B machine M, then the safety property ϕ must hold for the group protocol model $\mathbb{G}$. Formally, $(M \models I) \;\Rightarrow\; (\mathbb{G} \models \;\phi)$. In terms of equivalence between the two models, we can say that a protocol Model $\mathbb{G}$ is equivalent to an event-B model M, with regards to the security property, if the property ϕ holds in the model $\mathbb{G}$, and the invariant I holds in the model M. To illustrate this equivalence we need to show that $I \Rightarrow \phi$. Therefore, it is enough to show that the invariant I, with regard to M, implies the safety property ϕ, with regard to $\mathbb{G}$.

Theorem 6.3.1 *Secrecy Soundness.*

A group protocol, G, satisfies its secrecy property, ϕ, if there is an equivalent event-B model, M, that satisfies an event-B invariant, I, that implies the property ϕ. More formally, for the definition $(G \triangleq M)$, let $(M \models I)$, and $(I \Rightarrow \phi)$ be correct lemmas, then,

$$((G \triangleq M) \wedge (M \models I) \wedge (I \Rightarrow \phi) \Rightarrow (G \models \phi).$$

The proof is divided into two parts, we assume that lemmas are correct, then we proof the theorem based on that. In the next stage we proof each lemma separately.

Proof. Given $(G \triangleq M)$, $(M \models I)$, and $(I \Rightarrow \phi)$, we can deduce

$(M \models I) \wedge (I \Rightarrow \phi) \Rightarrow (M \models \phi)$

$(G \triangleq M) \wedge (M \models \phi) \Rightarrow (G \models \phi)$

□

We first define the equivalence relation between G and M, then we proof the lemma $(I \Rightarrow \phi)$. The lemma $(M \models I)$ is assumed to be correct in the event-B tool.

Definition 6.3.1 *A group protocol model G, is equivalent to an event-B model, M under certain conditions and semantically correct map from G to M. $G \triangleq M$ is defined as follows:*

For every component and condition in G there is an equivalent one in M. A protocol model is composed of $\mathbb{M}, \mathbb{S}, \mathbb{K}, \mathbb{K}_0, \mathbb{E}, \phi$, we map each component in G into an equivalent one in M.

Messages sets are mapped into an event-B variables by defining v over the set $\mathbb{M}$, and messages sets relations are mapped to event-B constants properties. $P(v)$ is a function of $\mathbb{M}$. These relations include the predicates about sets that should always hold.

Messages generation conditions are mapped into events guards. These conditions include predicates that should hold prior to executing an event, like having the appropriate key to encrypt or decrypt a message.

$G(v) = condition(\mathbb{M}, \mathbb{E})$

The secrecy property, ϕ, is mapped into an event-B invariant, I. This map is defined in lemma 6.3.1.

An event in $\mathbb{E}$ is mapped into event R, an event, $e_c \in \mathbb{E}$ with a precondition $condition$, where $m' = e_c(m_1, m_2, ...)$, the message generated from executing the even, can be defined concretely using an event-B $SELECT$ statement.

$Nameevent =$

ANYm$WHERE$

$condition$

$THEN$

R((m, m')

This map defines the relation $G \triangleq M$.

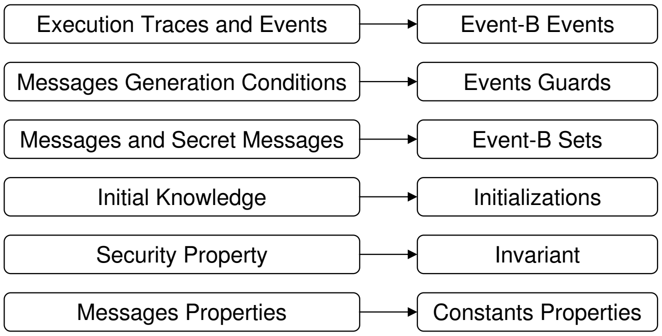


Figure 6.2: Mapping protocol primitives into event-B

To show that $I \Rightarrow \phi$, we need to establish a well-formed link between event-B invariant and the safety property. We split this formal link into two parts: the first deals with the initialization, and the second deals with executing the events. First we relate messages in $\mathbb{G}$ to variables in M. In Figure 6.2, we describe a map from public messages and secret messages to event-B sets and a map from messages sets relations to event-B constants properties. This map relates the variable m over the set of messages $\mathbb{M}$ directly to the variable v over event-B carrier sets and constants. The semantical correspondence between the variable m and the variable v is defined by this map.

We define the invariant I as $I = I_{init} \wedge I_E$, where I_{init} is the invariant predicate under the initial conditions, and I_E is the invariant predicate under executed events. Similarly, we define the safety property $\phi = \phi_{init} \wedge \phi_E$.

Lemma 6.3.1 $(I \Rightarrow \phi) = ((I_{init} \Rightarrow \phi_{init}) \ \wedge \ (I_E \phi \Rightarrow \phi_E))$

Proof. We define the well-formed conditions that guarantee the correctness of this Lemma in two steps, we first show that $(I_{init} \Rightarrow \phi_{init})$. We identify the initial events and initial set of messages in $\mathbb{G}$ under which the formula $(I_{init} \Rightarrow \phi_{init})$ holds. Then we define the predicates $P, I, G,$ and R presented in Lemmas 2.3.1 and 2.3.2 for the protocol model $\mathbb{G}$ such that Lemma 6.3.1 holds.
The definition of the group key protocol must satisfy the initial soundness conditions: $\mathbb{K}_0 \cap \mathbb{S} = \emptyset$ and $\forall e_i \in \mathbb{E}_i . m' := e_i(m) \Rightarrow m' \notin \mathbb{S}$, where e_i is an initial event that can be applied on the intruder's initial set of messages. We choose $R_I = \mathbb{E}_0$ to be the set of events that can be executed on $\mathbb{K}_0$.
We will define the constants property P and the initialization predicate R_I for the model $\mathbb{G}$ that will satisfy Lemmas 2.3.3 and 2.3.4. Then we define P, R, the predicate guards G, and the invariant I for the model $\mathbb{G}$ that will satisfy Lemmas 2.3.1 and 2.3.2.

Case 1 $(I_{init} \Rightarrow \phi_{init})$

- $P(m) = (\mathbb{K}_0 \neq \emptyset) \wedge (\mathbb{K}_0 \subset \mathbb{M}) \wedge (\mathbb{K} = \mathbb{K}_0)$
- $R_I = (e_i \in \mathbb{E}) \wedge (\exists (m' \in \mathbb{M}, m \in \mathbb{K}_0) \cdot m' := e_i(m))$
- $I(m) = m \in \mathbb{K}_0 \Rightarrow m \notin \mathbb{S}$

The message generation event $m' := e_i(m)$ is equivalent to the transition relation $R_I(v, v')$. This yields the formula $P(m) \Rightarrow \exists e_i \in \mathbb{E}_i \cdot m' := e_i(m)$ which is exactly Lemma 2.3.3 considering that $R_I = e_i$.

The invariant definition for the model $\mathbb{G}$ is $I(m) = m \in \mathbb{K} \Rightarrow m \notin \mathbb{S}$. We need to show that the invariant I holds for both $I(m)$ and $I(m')$. Since the protocol is initially sound, then both $I(m)$ and $I(m')$ hold by the fact that $\mathbb{K}_0 \cap \mathbb{S} = \emptyset$ and that the initial events cannot generate secret messages in S. If $m' := e_i(m)$ then $m' \notin \mathbb{S}$. Therefore we can write $(P(m) \wedge (m' := e_i(m))) \Rightarrow I(m')$, which is corresponds to Lemma 2.3.3 considering that $R_I = e_i$.

Case 2 $(I_E \Rightarrow \phi_E)$

- $P(m) = (\mathbb{K} \subset \mathbb{M})$
- $I(m) = (m \in \mathbb{K} \Rightarrow m \notin \mathbb{S})$
- $G(m) = ((\{m\}_k := \mathit{encr}(m, k) \Rightarrow k \in \mathbb{K}) \wedge (m := \mathit{decr}(\{m\}_k, k) \Rightarrow m \in \mathbb{K}))$
- $R = (e \in \mathbb{E}) \wedge (\exists m \in \mathbb{K}, m' \in \mathbb{M} \cdot m' = e(m))$

This message generation event is equivalent to the transition relation $R(v, v')$. Therefore, applying the predicates P, I, and G will lead to the relation R. We can write the formula $P(m) \wedge I(m) \wedge G(m) \Rightarrow \exists e \in \mathbb{E} \cdot m' = e_i(m)$ which is equivalent to Lemma 2.3.1 considering that the relation R is equivalent to an existing event $e \in \mathbb{E}$.

$\square$

The validity of the invariant $I(m')$ for the model $\mathbb{G}$ is expressed by the validity of the predicates $P, I, R,$ and G, where $m' := e(m)$. This can be written as $I(m) \wedge P(m) \wedge G(m) \wedge R \Rightarrow I(m')$, which corresponds to Lemma 2.3.2.

Under these conditions, we guarantee that when the invariant holds in event-B model, the secrecy property definition holds for the group key protocol model. These predicates should be considered carefully when providing the event-B implementation. Properties that can be expressed as invariants are verified using the translation process and event-B tool.

This completes the proof of Theorem 6.3.1. The major restriction on this method is that it can reason about the execution of a single protocol event, i.e. join or leave. However, this is enough to model and verify group secrecy when a member joins or leaves the group.

6.4 Verification of Forward Secrecy in Event-B Refinement

In previous section, a well-formed link between the semantics of the group protocol and event-B models is established. Then we showed that when the invariant I holds for event-B machine M, the safety property ϕ must also hold for the group protocol model $\mathbb{G}$. The verification methodology for forward secrecy is built on top of the methodology we use for secrecy.

The TGDH protocol designers [49] defined forward secrecy as follows: forward secrecy guarantees that a passive adversary who knows a contiguous subset of old group keys (say $\{K_0, K_1, \ldots, K_i\}g$) cannot discover any subsequent group key K_j for all i and j , where $j > i$. We will follow this definition when we consider the verification of the protocol designed in [49]. However, this is different from the definition presented in chapter 3.

Figure 6.3 below shows the necessary modifications needed. We will consider the model M as an abstract one, and defined on top of that, a refined model, M_c, and a gluing invariant J linking variables of the abstract model to those of the concrete or refined one (M_c). In addition to the previous map defined from G to M, we will use the variable v_c to represent the est of intruders message in the refined model. Therefore, $J(v, v_c)$ will represent the gluing invariant which represents forward secrecy property before the relation R_c. In addition, we use $I(v_c)$ to represent the invariant in the refined model, which corresponds to secrecy property of the refined protocol model, G_c. The relation R_c will be defined the same way as the relation R.

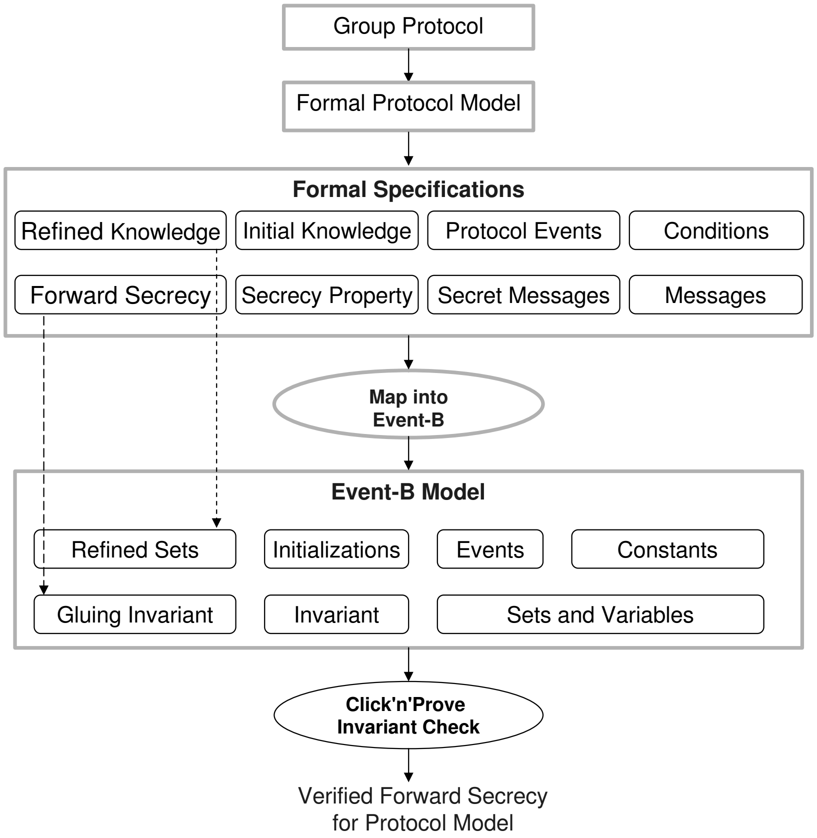


Figure 6.3: Refined Event-B Method for Forward Secrecy

In event-B refinement can be done with events or variable. In our case, the group protocol join or leave events have the same semantics in both secrecy and forward secrecy, therefore, it will have the same definition in both the abstract and refined event-B models, i.e., $R_c = R$. We use $R(v, v')$ to represent join or leave events, which updates the intruder's set of knowledge, the variable v here. In the refined model, we will use the same relation, we call it $R_c(v_c, v'_c)$, that will update the intruder's set of knowledge, v_c, in the refined model M_C.

The correctness of forward secrecy, ϕ_f, with regards to the event-B concrete model M_c is achieved through the correctness of the gluing invariant $J(v', v'_c)$. Figure 6.4 below illustrates the link between the abstract and refined model to achieve a model for forward secrecy in event-B.

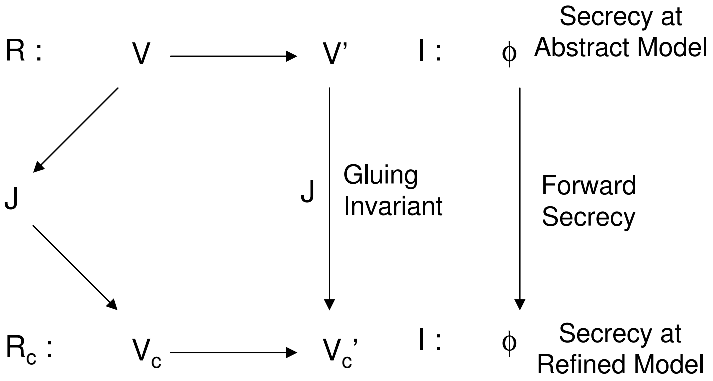


Figure 6.4: Relationship between Abstract and Refined Models

Theorem 6.4.1 *Forward Secrecy Soundness.*

A group protocol, G_c, satisfies its secrecy property, ϕ_f, if there is an equivalent event-B model, M, that satisfies an event-B invariant, I, and a refined event-B model M_c that satisfies an event-B invariant I, and a gluing variable $J(v, v_c)$ that implies ϕ_f in the existence of a relation $R_c(V_c, V_{c'})$.

Formally, let $(G_c \triangleq M_c)$, $(I_c \Rightarrow \phi_f)$, $(M \models I)$, and $(M_c \models I_c)$ be correct lemmas, then

$$(G_c \triangleq M_c) \wedge (M_c \models I_c) \wedge (J \Rightarrow \phi_f) \Rightarrow (G_c \models \phi_f).$$

Proof. Assuming $(G_c \triangleq M_c)$, $(J \Rightarrow \phi_f)$, and $(M_c \models J)$, we can deduce:

$(M_c \models J) \wedge (J \Rightarrow \phi_f) \Rightarrow M_c \models \phi_f$.

$(G_c \triangleq M_c) \wedge M_c \models \phi_f \Rightarrow (G_c \models \phi_f)$.

□

Event-B invariant cannot reason about backward secrecy because invariant cannot be used in a reverse manner, i.e., refining the intruder's knowledge back in time. In backward secrecy the intruder is assumed to be an active user in the group while trying to discover older secret shares prior to his membership. Therefore, refinement of secrecy can only be used for forward secrecy. Based on this, key independence (collusion) cannot be modeled in this method as is.

6.5 Application: Secrecy in TGDH Protocol

In this section, we apply the approach on a group key protocol that generates a key in a distrusted group. We show how the conditions defined for the correctness of the above model can be concretely applied on a real protocol. The intended secrecy property, along with its conditions, are efficiently defined and checked as event-B invariant.

We first introduce the basic Tree-based Group Diffie-Hellman protocol (TGDH) as it is designed in [49]. All TGDH protocols have the following features:

- Each group member contributes an equal share to the group key, and the key is a function of all current group members shares.
- The share of each member is secret and is never revealed.
- When a new member joins the group, one of the old members changes its share, and new members' shares are factored into the group key.
- When an existing member leaves the group, its share is removed from the new group key, and at least one remaining member changes its key share.

- All protocol messages are signed, time-stamped, sequence-numbered, and type-identified by the sender.

After every membership change, all remaining members independently update the key tree structure and recompute identical key trees after any membership event. A group key can be computed from any members secret share and all blind keys on the co-path to the root. Blind keys are the siblings of the nodes on the key path. The members own secret share and all sibling blind keys on the path to the root enable a member to compute all intermediate keys on its key-path, including the root group key. Figure 6.5 shows a binary tree structure that represents the group members, their own secret shares, and the secret sub-keys on every node up to the root. As part of the protocol, a group member can take on a special sponsor role, which involves computing intermediate keys and broadcasting to the group. Each broadcasted message contains the senders view of the key tree, which contains each blind key known to the sender [49].

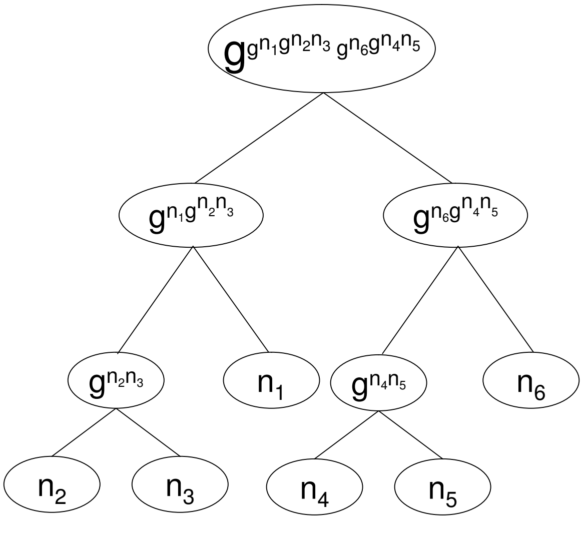


Figure 6.5: Tree-based GDH Protocol Binary Tree Structure

The group key is calculated by each member based on his/her key-path and blind keys. For instance, for a member M_3 at node n_3, the key-path is the set of messages

$\{n_3,\ g^{n_2 n_3},\ g^{n_1 g^{n_2 n_3}}\}$. The set of blind keys ordered as they appear up to the root is $\{\ g^{n_2},\ g^{n_1},\ g^{g^{n_6 g^{n_4 n_5}}}\}$. The group key at the root is calculated directly using the two sets:

$$GroupKey = g^{g^{n_1 g^{n_2 n_3}} g^{n_6 g^{n_4 n_5}}}$$

The protocol designers presented four types of security properties: *group key secrecy*, which guarantees that it is computationally infeasible for a passive adversary to discover any group key, intuitively, that the attacker should not be able to obtain a key that honest users think to be safe; *forward secrecy* guarantees that a passive adversary who knows a contiguous subset of old group keys cannot discover any subsequent group key; *backward secrecy*, which guarantees that a passive adversary who knows a contiguous subset group keys cannot discover preceding group key, and finally, *key independence*, which guarantees that a passive adversary who knows a proper subset of group keys cannot discover any other group key. The authors of [49] provided an informal proof that their protocol satisfies these security property. In this work, we provide a formal proof for group key secrecy property under certain conditions. This property can be described as a *correct key construction property*, which guarantees that only group members, who are of knowledge to their own private shares, can calculate the group key at root. On the other hand, an adversary, who has knowledge to all blind sub-keys cannot find a full path to calculate the root key.

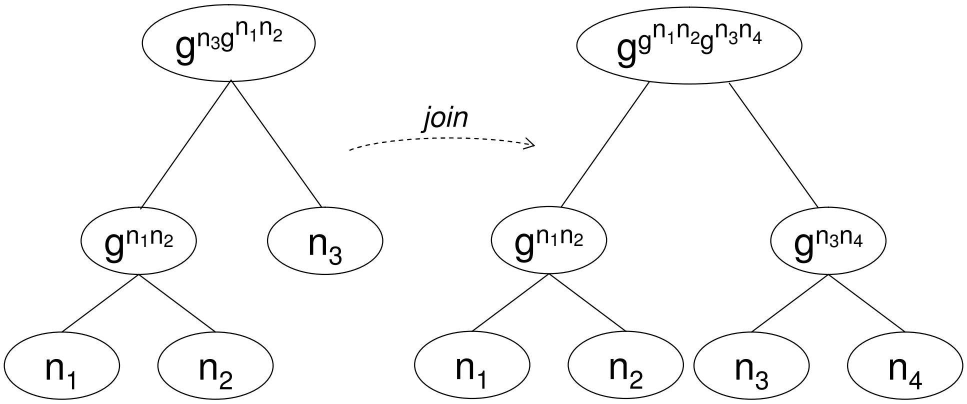


Figure 6.6: Join Event in the TGDH Protocol

We illustrate our method on a group protocol composed of three members, then we apply a join event for a fourth member. Figure 6.6 shows the modification on the tree structure when a new member joins the group, we define the group protocol components before and after this event takes place. Assuming that a passive adversary is monitoring the group activity, the knowledge set is built based on the blind keys interchanged between members. Based on this configuration, we show all group protocol components, including secrecy property, and the equivalent event-B model including the invariant, before the join event takes place:

$$\mathbb{M} = \{n_1,\ n_2,\ n_3,\ g^{n_1},\ g^{n_2},\ g^{n_3},\ g^{n_1 n_2},\ g^{g^{n_1 n_2}},\ g^{n_3 g^{n_1 n_2}}\}$$

$$\mathbb{S} = \{n_1,\ n_2,\ n_3,\ g^{n_1 n_2},\ g^{n_3 g^{n_1 n_2}}\}$$

$$\mathbb{K}_0 = \{n_i,\ g^{n_i}\}$$

$$\mathbb{K} = \{n_i,\ g^{n_i},\ g^{n_1},\ g^{n_2},\ g^{n_3}, g^{g^{n_1 n_2}}\}$$

$$GroupKey = g^{n_3 g^{n_1 n_2}}$$

Then, we show the same components after the join event of a new member with a new secret contribution n_4. Note that group key secrecy has the same definition and should be valid always, before and after a join (or leave) event takes place.

$$\mathbb{M} = \{n_1,\ n_2,\ n_3,\ n_4, g^{n_1},\ g^{n_2},\ g^{n_3},\ g^{n_4},\ g^{n_1 n_2},\ g^{n_3 n_4},\\ g^{g^{n_1 n_2}},\ g^{g^{n_3 n_4}},\ g^{g^{n_1 n_2} g^{n_3 n_4}}\}$$

$$\mathbb{S} = \{n_1,\ n_2,\ n_3,\ n_4,\ g^{n_1 n_2},\ g^{n_3 n_4},\ g^{g^{n_1 n_2} g^{n_3 n_4}}\}$$

$$\mathbb{K} = \{n_i, g^{n_i},\ g^{n_1},\ g^{n_2},\ g^{n_3},\ g^{n_4},\ g^{g^{n_1 n_2}},\ g^{g^{n_3 n_4}}\}$$

$$GroupKey = g^{g^{n_1 n_2} g^{n_3 n_4}}$$

$$\phi = GroupKey \notin \mathbb{K} \wedge \mathbb{K} \cup \mathbb{S} = \emptyset$$

6.5.1 Secrecy Model in Event-B Invariant

The event-B model for the protocol components is described below. We first define the event-B sets for blind keys ($BLINDKEYS$), the general set of messages (MS), the intruder's set of messages (K), and the set of secret keys (S). Then we define a number of variables over the above sets of messages. We describe the current status of the group by initializations where each of the above sets is concretely defined. The secrecy property is defined as an invariant that combines a set of conditions to be satisfied at the initialization and after executing the event: $K \cap S = \emptyset$. Some of the protocol characterizes can also be encoded within this invariant, such as $K \subset MS \wedge S \subset M$. We also define an event to represent the protocol action (join/leave).

The the TGDH protocol components are defined in Click'n'Prove tool as follows:

```
SYSTEM TGDHProtocol
SETS
  $BLINDKEYS$ /* set of Blind keys */
  $MS$; /* set of messages */
  $K$ /* Intruder's set of knowledge*/
  $S$ /* Set of secret messages */
VARIABLES
  $intruderKey, msgBefore, msgAfter, bk, Gkey$
INVARIANT
/* malicious participant cannot evaluate to GK */
  $K \cap S = \emptyset \wedge GKey \notin K \wedge K \subset MS \wedge S \subset MS \ldots$
INITIALISATION
  $BLINDKEYS := \{g^{N_1}, g^{N_2}, g^{N_3}, g^{g^{N_1 N_2}}, \ldots\};$
  $MS := N_1, g^{N_1}, N_2, g^{N_2}, \ldots\};$
  $K := g^{N_1}, g^{N_2}, \ldots$
  $S := \{N_1, N_2, g^{n_1 n_2}, g^{n_1 n_2 g^{n_3}}, \ldots\}$
EVENTS  $eventB\_tgdh \triangleq\ \ldots$ /*for a protocol event*/
END
```

In the above event-B model, the sets of messages MS, K, and S, are directly defined from the above sets $\mathbb{M}, \mathbb{K}$, and $\mathbb{S}$, respectively. The group key has basically the same definition, and secrecy property is defined as an event-B invariant that contains, in addition to group key secrecy, certain conditions on messages sets to insure the consistency of the map,

$(K \cap S = \emptyset) \wedge GKey \notin K \wedge K \subset MS \wedge S \subset MS.$ To be consistent with the group structure, we also defined the set of blind keys in event-B as follows:

$BLINDKEYS = \{g^{N_1}, g^{N_2}, g^{N_3}, g^{g^{N_1N_2}}, \dots\};$

An event-B definition that captures the behavioral semantics of a join event which will result in updating the intruder's set of knowledge is described in Click'n'Prove tool as follows:

```
eventB_tgdh $\triangleq$ /* for any message m */
ANY $msgBefore, msgAfter, bk, \dots$ WHERE
  $msgBefore \in K \wedge msgAfter = g^{g^{N_3N_4}} \wedge \dots$
THEN
/* update intruder's set of messages after executing the event */
  $GKey := g^{g^{N_1N_2}g^{N_3N_4}}$ /* calculate group key */
$K := K \cap msgAfter$
/* update the set of secret messages */
  $S := S \cup \{Gkey, N_4, g^{N_3N_4}, \dots\}$
END
```

After this event is executed in the tool, new blind keys will be generated and added to that set. The new secret group key is calculated based on the new contribution of the joined member, n_4.

$GKey = g^{g^{n_1n_2}g^{n_3n_4}}$

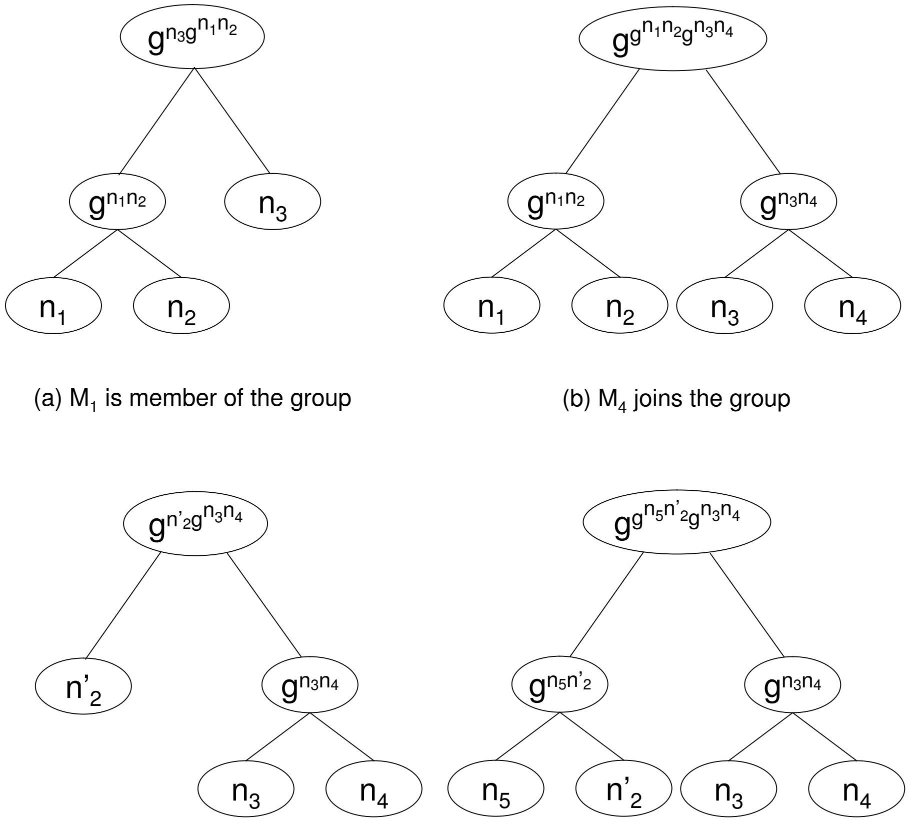
In Click'n'Prove implementation, we first consider the static case of key construction under the assumption that basic DH key construction (on tree leaf nodes) is correct. We then consider the dynamic case by applying events such as join and leave and verify the correctness of key construction for a bounded tree size and bounded number of events. The event-B invariant has been proven totally. The number of generated proof obligations are three, all proof obligations are proven automatically, and then the initial model of the group key protocol is validated. The event-B invariant, I, defined in Click'n'Prove model above, implies the group protocol secrecy semantically, $I \Rightarrow \phi$. The event-B tool guarantees that $M \models I$. We have shown in the previous section that the group protocol G is mapped into an event-B model M. Therefore we can conclude the correctness of the secrecy property ϕ for the protocol model G, $G \models \phi$.

The proposed solution allows us to verify the required property, however, one limitation of our approach is related to the fact that event-B operations are defined only over finite sets. Therefore, a bounded number of participants and protocol events should be applied. Another limitation is due to the fact that we verify the property under the execution of a single event. However, this approach is sufficient for the target property, where key distribution is abstracted away because we are concerned only with modeling key construction but not key distribution or authentication property.

In addition, to modeling the relationship between the secret keys and blind keys an exponent operator is needed, therefore, the set of possible blinded keys is directly related to the number of participants represented by the tree level, i.e, the model size in event-B is directly related to the number of participants. Hence, a huge set of keys should be modeled, where the automatic generation of these keys is infeasible because no exponent operator is supported by event-B. Therefore, applying invariant checking becomes limited by the issue of generating this set manually. Even though there are some limitations for the approach, event-B can be used in modeling specific protocols behaviors, like key construction, and tree-based protocol primitives can be modeled directly in event-B for safety properties verification.

6.5.2 Forward Secrecy Model in Event-B Refinement

We illustrate our method on a group protocol composed of three members, then we apply a join event for a fourth member. Figure 6.7 shows the modification on the tree structure when a new member joins the group, we define the group protocol components before and after this event takes place. Assuming that a passive adversary is monitoring the group activity, the knowledge set is built based on the blind keys interchanged between members. Based on this configuration, we show all group protocol components, including secrecy property, and the equivalent event-B model including the invariant, before join event takes place:



(a) M_1 is member of the group

(b) M_4 joins the group

(c) M_1 leaves the group

(d) M_5 joins the group

Figure 6.7: Forward Secrecy in the TGDH Protocol

In Figure 6.7 (a), the current set of messages, secret set of messages, and the current group key are defined as follows:

$\mathbb{M} = \{n_1,\ n_2,\ n_3,\ g^{n_1},\ g^{n_2},\ g^{n_3},\ g^{n_1 n_2},\ g^{g^{n_1 n_2}},\ g^{n_3 g^{n_1 n_2}}\}$

$\mathbb{S} = \{n_1,\ n_2,\ n_3,\ g^{n_1 n_2},\ g^{n_3 g^{n_1 n_2}}\}$

$GroupKey_a = g^{n_3 g^{n_1 n_2}}$

Then, after member M_4 joins the group, as shown in Figure 6.7 (b), the above variables become:

$$\mathbb{M} = \{n_1,\ n_2,\ n_3,\ n_4, g^{n_1},\ g^{n_2},\ g^{n_3},\ g^{n_4},\ g^{n_1n_2},\ g^{n_3n_4}, g^{g^{n_1n_2}},\ g^{g^{n_3n_4}},\ g^{n_3g^{n_1n_2}},$$
$$g^{g^{n_1n_2}g^{n_3n_4}}\}$$

$$\mathbb{S} = \{n_1,\ n_2,\ n_3,\ n_4,\ g^{n_1n_2},\ g^{n_3n_4},\ g^{g^{n_1n_2}g^{n_3n_4}}\}$$

$$GroupKey_b = g^{g^{n_1n_2}g^{n_3n_4}}$$

The next step, is that we let member M_1, who will be assumed to be dishonest later, leave the group, where a new secret share n_2' is generated as in Figure 6.7 (c). The above variables become:

$$\mathbb{M} = \{n_1,\ n_2,\ n_2',\ n_3,\ n_4, g^{n_1},\ g^{n_2},\ g^{n_2'},\ g^{n_3},\ g^{n_4},\ g^{n_1n_2},\ g^{n_3n_4},\ g^{g^{n_1n_2}},\ g^{g^{n_3n_4}},$$
$$g^{n_3g^{n_1n_2}},\ g^{g^{n_1n_2}g^{n_3n_4}},\ g^{n_2'g^{n_3n_4}}\}$$

$$\mathbb{S} = \{n_2',\ n_3,\ n_4,\ g^{n_3n_4},\ g^{n_2'g^{n_3n_4}}\}$$

$$GroupKey_c = g^{n_2'g^{n_3n_4}}$$

Finally, this is the join (or similarly leave) event on which we will check invariant for the refined model. A new member, M_4 joins the group as in Figure 6.7 (d), and we assume that M_1 is monitoring the group. We first illustrate secrecy first, then forward secrecy:

$$\mathbb{M} = \{n_1,\ n_2,\ n_2',\ n_3,\ n_4,\ n_5, g^{n_1},\ g^{n_2},\ g^{n_2'},\ g^{n_3},\ g^{n_4},\ g^{n_5},\ g^{n_1n_2},\ g^{n_3n_4},\ g^{n_5n_2'},$$
$$g^{g^{n_1n_2}},\ g^{g^{n_3n_4}},\ g^{g^{n_5n_2'}},\ g^{n_3g^{n_1n_2}},\ g^{g^{n_1n_2}g^{n_3n_4}},\ g^{g^{n_5n_2'}g^{n_3n_4}},\ g^{n_2'g^{n_3n_4}}\}$$

$$\mathbb{S} = \{n_2',\ n_3,\ n_4,\ n_5,\ g^{n_3n_4},\ g^{n_5n_2'},\ g^{n_2'g^{n_3n_4}},\ g^{g^{n_5n_2'}g^{n_3n_4}}\}$$

$$GroupKey_d = g^{g^{n_5n_2'}g^{n_3n_4}}$$

We define the set $\mathbb{K}$, as viewed by a member monitoring the group from outside, without being a member at any previous time. Before M_5 joins the group:

$$\mathbb{K} = \{n_i, g^{n_i},\ g^{n_1},\ g^{n_2},\ g^{n_2'},\ g^{n_3},\ g^{n_4},\ g^{g^{n_1n_2}},\ g^{g^{n_3n_4}}\}$$

Then we show the set of messages of member M_5 joins the group:

$$\mathbb{K} = \{n_i, g^{n_i},\ g^{n_1},\ g^{n_2},\ g^{n_2'},\ g^{n_3},\ g^{n_4},\ g^{n_5},\ g^{g^{n_1n_2}},\ g^{g^{n_3n_4}},\ g^{g^{n_5n_2'}}\}$$

Secrecy implies that the intruder monitoring the group should not be able to calculate the group key $GroupKey_5$, (or any secret share or sub-key).

In forward secrecy the set of messages $\mathbb{K}$ is refined with the knowledge gained by user M_1 while member in the group, and is defined as:

$\mathbb{K}' = \{n_i,\ n_1,\ g^{n_i},\ g^{n_1},\ g^{n_2},\ g^{n_2'},\ g^{n_3},\ g^{n_4},\ g^{n_5}, g^{n_1 n_2},\ g^{g^{n_1 n_2}},\ g^{g^{n_3 n_4}},\ g^{g^{n_5 n_2'}},\ g^{n_3 g^{n_1 n_2}}\}$

$GroupKey = g^{g^{n_1 n_2} g^{n_3 n_4}}$

$\phi_f = GroupKey_d \notin \mathbb{K}' \wedge \mathbb{K}' \cup \mathbb{S} = \emptyset$

6.6 Summary

The main objective of this chapter was to provide automated invariant checking for group key secrecy and forward secrecy properties. This was not feasible in neither the rank functions, nor the inference system methods. We used event-B invariants to model and verify group key secrecy, then, on top of this, we used event-B refinement to model and verify forward secrecy. For this purpose, a formal link between the semantics of group protocols model and event-B was established. The result was combining event-B and group protocol model to be able to use specific features in event-B to model protocol actions and verify the required property. However, we restrict the group protocol model to be verified to certain conditions in order to guarantee the correctness of the method and the applicability of first-order logic theorem proving. This includes the number of participants, abstracting the exponentiation operator for Diffie-Hellman style protocols, and finally, applying a single protocol event.

We applied this approach on a group key protocol, the tree based Group Diffie-Hellman protocol and provided invariant checking for secrecy under the static and the dynamic case by applying a single event (join/leave). In contrast to our work, the authors of the TGDH protocol [49] provided an informal, non-intuitive and simple proof for secrecy property.

As a limitation of the approach, only invariant properties can be modeled and verified. This is due to the target model and verification tool, namely, event-B and Click'n'Prove tool. However, invariant checking is adequate to model many security properties. In addition, we were able to conduct invariant checking for a limited number of tree levels due to the lack of exponent operator in the prover.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Cryptographic protocols are used for establishing secure communication. In particular, group key protocols allow group members to exchange or establish keys to encrypt and authenticate messages within the group. After forming a group, members can encrypt, decrypt, and authenticate messages to and from other members of the group. Provided secure underlying cryptography, anyone outside of the group cannot eavesdrop on the communication or inject a message that will successfully authenticate

The correctness of group key protocols in communication systems remains a great challenge because of the sensitivity of the services provided. The verification problem for group key protocols is more challenging because properties for these protocols are not trivial extensions of the two-parties models.

Many approaches in the literature analyze group key protocols by extending methods developed for two-parties protocols, such as the works of Crazzolara and Winskel [26, 27]. Other methods analyze the algebraic aspects of the protocol like the works of Mazaré [57], Millen and Shmatikov [62], and Verma *et al.* [84]. Other methods used theorem proving to prove authentication in GDH protocol [29], used PVS to verify the proper group agreement in Enclaves protocol [53] , or used the Isabelle theorem prover for the verification of the Kerberos authentication system [11] and Secure Electronic Transaction (SET) protocol [10]. These approaches were unable to reason about group key secrecy from the non-algebraic point of view, which can be a source of attacks for these protocols. Yet, to reason about forward and backward secrecy is another problem.

In this thesis, we describe a framework for the formal specification and verification of group key management protocols. The framework is based on three complementary approaches: a rank theorems based approach, a rank functions based inference system, and an event-B first-order theorem proving approach.

In the first two methods we adapted the idea of rank functions introduced by Schneider *et al.* [35, 75] in order to be able to verify security properties for group oriented protocols. A set of sound rank functions that satisfy specific requirements is defined. Then, this set of rank functions is used in two directions, in the first to define rank theorems that guarantee the correctness of the security property, and in the second we define an inference system that is composed of a set of inference rules over rank functions, where every rule can be applied in order to generate new knowledge and assign new ranks to these generated messages. A special rule called *Attack* is defined to represent the bottom of the system, and when executed, illustrates an attack in the protocol. This way, reasoning about absence of attacks is possible in the rank theorems, and reasoning about their existence is possible using the inference system.

To overcome the heavy cost of interactive theorem proving as used in above two methods, a third method has been proposed, where event-B first order theorem prover is used to verify secrecy property as an event-B invariant. This is not a straightforward task, and is based on a correct semantical link between the two models. Forward secrecy is modeled in event-B using refinement on top of secrecy.

The above methods were applied on different protocols from the literature. First we applied the rank theorem proof environment on the Enclaves protocol in order to verify related forward secrecy and backward secrecy in PVS. Then we implemented the verification of a generic Diffie-Hellman group protocol in PVS based on the inference system approach. Finally, We applied the event-B based approach on a Tree based Group Diffie-Hellman protocol and used invariant checking to verify the correctness of key construction.

The reason why we applied every method on a different protocol is to compromise between the complexity of the protocol with the efficiency of the method. The rank theorem method is efficient in modeling complex features of the protocols since it is based on a higher-order logic model, therefore, it is appropriate to choose a protocol with a complex group key management procedure such as Enclaves. On the other hand, the event-B based method is based on first-order logic, therefore, a protocol with simple key management is preferable, such as TGDH protocol, where this method can perform better. To illustrate the efficiency of the rank functions based inference system, a protocol with a known attack was used.

This way, we were able to address security properties for group key protocols that were not addressed formally before. We do not claim that our framework can deal with all possible security protocols, or every group protocol. Each of the methods presented above has its shortcomings and limitations. The major limitation of the rank theorems method is in the implementation of the rank theorems, which requires a considerable amount of effort and time in theorem proving. The rank theorem is defined at high level of abstraction in order to capture the rank function and the security property requirements. Therefore, when implemented in a theorem proving tool, it requires a lot of effort to be proven. In addition, it can only reason about absence of attacks.

The limitation of the inference system is again the implementation and proof interactivity. Besides, it can only efficiently reason about the existence of attacks. As a limitation of the event-B semantics based approach, only invariant properties can be modeled and verified. This is due to the target model and verification tool, namely, event-B and Click'n'Prove. However, invariant checking is adequate to model many security properties. In addition, we were able to conduct invariant checking for a limited number of tree levels due to the lack of exponent operator in the prover.

The event-B semantics based method major limitations is the inability to reason about backward secrecy and key independence. That is because group key secrecy refinement results in forward secrecy, while, for backward secrecy, an opposite refinement in time is required which is infeasible.

We believe we presented a promising approach to the verification of group key protocols. A significant advantage of our framework is its applicability on different types of group protocols, and in addition, the ability to model and verify various types of group secrecy properties. However, for the verification a considerable amount of reasoning is needed to prove that the group protocols satisfy the intended security properties. The exception here is the event-B semantics based method, once the properties and the basic model are established, using the invariant checking to deduce the validity of security properties of the group protocol is essentially straightforward. In addition, the methods we present here are considered complementary to cryptographic analysis techniques that are used to verify group key protocols, since treating the protocol at the higher level of abstraction is done at a lower cost.

Another interesting feature of our framework is that the theorem statements are not strongly connected to the underlying semantics. They are therefore in a sense independent of the semantics. Indeed, we believe the framework could be transferred to any other semantics powerful enough to express at least the notion of independence and the security properties, and which has a similar execution model. In general, our theorems are tight in the sense that if any precondition is not satisfied, the theorem is no longer true.

7.2 Open Issues and Future Work

The application of the inference system on group protocols in the existence of a passive adversary should be investigated further. Another direction is to provide an implementation of the inference system itself, rather than defining it in a theorem prover. This will provide more flexibility for modeling different protocols. However, if we implement our inference system, then we need to implement some strategies in order to guarantee the success of the verification process. If necessary the user can help the system in order to find an attack.

The inference system can reason about absence of attacks in protocols using the Completeness Theorem 5.2.2. An *indirect* proof can be generated using the completeness theorem by proving that the strategy is fair and the application of our inference system diverges. This *indirect* proof can be achieved by generating partial proofs that affirm that the inference system will diverge when the applied strategy is fair. We believe that in order to be apply this approach, we have to provide an implementation for the inference system that allows partial proofs based on divergence. This later issue will be considered for further study.

In the event-B based method, only invariant properties can be modeled and verified. This is due to the target model and verification tool, namely, event-B and Click'n'Prove tool. However, invariant checking is adequate to model properties that describe secrecy. An interesting issue to be considered in the future is the ability to model and verify liveness security properties using event-B. In addition, the event-B based model can be extended to handle parameterized number of participants. It will also be interesting to investigate the possibility of modeling backward secrecy and key independence using event-B. However, in order to achieve this, major modifications of the approach are required, mainly, tool support should be investigated more. Other group protocols from [55] or [74] can be formalized and verified using this approach.

Group key protocols have applications in wireless sensors networks and wireless networks. The verification problem in this case will be different because these networks are ad-hoc, and therefore the threat model will be different.

Methods for high-level reasoning can treat increasingly complex security protocols. Formal approaches suggest such high-level reasoning principles, and even permit automated proofs. In addition, some formal approaches capture naive but powerful intuitions about these protocols. Formal reasoning about computational and analysis methods will increase the appeal and accessibility of these methods [1]. However, to reason about analysis and computational aspects in higher-order logics, specific theorems are needed for computational complexity theory, algebraic computations schemes, and probabilistic theorems [42] that can distinguish encryption and algebraic operations.

Bibliography

[1] M. Abadi and P. Rogaway. Reconciling Two Views of Cryptography (The Computational Soundness of Formal Encryption). *Journal of Cryptology*, 20(3):395–395, 2007.

[2] J. Abrial. *The B-Book: Assigning Programs to Meanings*. Cambbridge University Press, 1996.

[3] J. Abrial. Extending B without Changing it (for Developing Distributed Systems). In *1st Conference on the B method, Putting into Practice Methods and Tools for Information System Design*, pages 169–190. Institut de Recherche en Informatique de Nantes, 1996.

[4] J. Abrial and D. Cansell. Click'n'Prove: Interactive Proofs within Set Theory. In *Theorem Proving in Higher Order Logics*, volume 2758 of *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2003.

[5] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed System*. John Wiley & Sons, 2008.

[6] S. Andova, C. Cremers, K. Gjøsteen, S. Mauw, S. F. Mjølsnes, and S. Radomirović. A Framework for Compositional Verification of Security Protocols. *Information and Computation*, 206(2-4):425–459, 2008.

[7] G. Ateniese, M. Steiner, and G. Tsudik. New Multiparty Authentication Services and Agreement Protocols. *IEEE Journal of Selected Areas in Communications*, 18(4):628–639, 2000.

[8] M. Baugher, B. Weis, T. Hardjono, and H. Harney. The Group Domain of Interpretation (GDOI). RFC 3547, Internet Engineering Task Force. Available at http://www.ietf.org/rfc/rfc3547.txt, July 2003.

[9] G. Bella. Inductive Verification of Smart Card Protocols. *Journal of Computer Security*, 11(1):87–132, 2003.

[10] G. Bella, F. Massacci, and L. Paulson. The Verification of an Industrial Payment Protocol: The SET Purchase Phase. In *ACM Conference on Computer and Communications Security*, pages 12–20. ACM Press, 2002.

[11] G. Bella and L. Paulson. Using Isabelle to Prove Properties of the Kerberos Authentication System. In *Workshop on Design and Formal Verification of Security Protocols. DIMACS*, 1997.

[12] N. Benaissa, D. Cansell, and D. Méry. Integration of Security Policy into System Modeling. In *Formal Specification and Development in B*, volume 4355 of *Lecture Notes in Computer Science*, pages 232–247. Springer-Verlag, January 2007.

[13] M. Boreale and M. Buscemi. Symbolic Analysis of Crypto-Protocols Based on Modular Exponentiation. In *Mathematical Foundations of Computer Science*, volume 2747 of *Lecture Notes in Computer Science*, pages 269–278. Springer-Verlag, 2003.

[14] E. Bresson, O. Chevassut, and D. Pointcheval. Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions. In *International Conference on the Theory*

and Applications of Cryptographic Techniques, Lecture Notes in Computer Science, pages 321–336. Springer-Verlag, 2002.

[15] E. Bresson, O. Chevassut, and D. Pointcheval. Provably-Secure Authenticated Group Diffie-Hellman Key Exchange. *ACM Transactions on Information and System Security*, 10(3), August 2007.

[16] E. Bresson and M. Manulis. Malicious Participants in Group Key Exchange: Key Control and Contributiveness in the Shadow of Trust. In *Autonomic and Trusted Computing Conference*, volume 4610 of *Lecture Notes in Computer Science*, pages 395–409. Springer-Verlag, July 2007.

[17] C. Brown. *Automated Reasoning in Higher-order Logic*. College Publications, 2007.

[18] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[19] M. Butler. On the Use of Data Refinement in the Development of Secure Communications Systems. *Formal Aspects of Computing*, 14(1):2–34, 2002.

[20] A. Mathuria C. Boyd. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003.

[21] CCITT. CCITT Draft Recommendation X.509. The Directory Authentication Framework, version 7, November 1987.

[22] I. Chatzigiannakis, E. Konstantinou, V. Liagkou, and P. Spirakis. Design, Analysis and Performance Evaluation of Group Key Establishment in Wireless Sensor Networks. *Electronic Notes in Theoritical Computer Science*, 171(1):17–31, 2007.

[23] E. Clarke, O. Grumberg, and D. Long. Verification Tools for Finite-State Concurrent Systems. In *A Decade of Concurrency, Reflections and Perspectives Workshop*, pages 124–175, London, UK, 1993. Springer-Verlag.

[24] E. M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[25] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[26] F. Crazzolara. *Language, Semantics, and Methods for Security Protocols*. PhD thesis, BRICS, Denmark, May 2003.

[27] F. Crazzolara and G. Winskel. Events in Security Protocols. In *ACM conference on Computer and Communications Security*, pages 96–105. ACM Press, 2001.

[28] C. Cremers and S. Mauw. Operational Semantics of Security Protocols. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *Lecture Notes in Computer Science*, pages 66–89. Springer-Verlag, 2005.

[29] R. Delicata and S. Schneider. A Formal Model of Diffie-Hellman using CSP and Rank Functions. Technical Report CSD-TR-03-05, Department of Computer Science, Royal Holloway, University of London, 2003.

[30] R. Delicata and S. Schneider. Temporal Rank Functions for Forward Secrecy. In *Computer Security Foundations Workshop*, pages 126–139, Washington DC, USA, June 2005. IEEE Computer Society Press.

[31] R. Delicata and S. Schneider. An Algebraic Approach to the Verification of a Class of Diffie-Hellman Protocols. *International Journal of Information Security*, 6(2):183–196, 2007.

[32] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

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

[34] B. Dutertre, H. Saïdi, and V. Stavridou. Intrusion-Tolerant Group Management in Enclaves. In *Dependable Systems and Networks*, volume 2467 of *Lecture Notes in Computer Science*, pages 213–216. Springer-Verlag, July 2001.

[35] 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.

[36] Wikipedia Encyclopedia. http://www.wikipedia.org/, 2008.

[37] F. Fábrega. Strand Spaces: Proving Security Protocols Correct. *IOS Journal of Computer Security*, 7(2-3):191–230, 1999.

[38] N. Ferguson and B. Schneier. A Cryptographic Evaluation of IPSec. Technical report, Counterpane Internet Security Inc., 2000.

[39] M. Fitting. *First-order Logic and Automated Theorem Proving*. Springer-Verlag, 1996.

[40] M. Gordon and T. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic.* Cambridge University Press, 1993.

[41] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409, Internet Engineering Task Force. Available at http://www.ietf.org/rfc/rfc2409, November 1998.

[42] O. Hasan. *Formal Probabilistic Analysis using Theorem Proving*. PhD thesis, Concordia University, Montreal, Canada, April 2008.

[43] P. Hoffman. The Internet Key Exchange version 1 (IKEv1). RFC 4109, Internet Engineering Task Force. Available at http://ietf.org/rfc/rfc4109, November 2005.

[44] G. Horng. Cryptanalysis of a Key Management Scheme for Secure Multicast Communications. *IEICE Transcations on Communication*, E85-B(5):1050–1051, 2002.

[45] J. Katz and J. Shin. Modeling Insider Attacks on Group Key-Exchange Protocols. In *ACM Conference on Computer and Communications Security*, pages 180–189. ACM Press, 2005.

[46] J. Katz and M. Yung. Scalable Protocols for Authenticated Group Key Exchange. In *Advances in Cryptology*, volume 2729 of *Lecture Notes in Computer Science*, pages 110–125. Springer-Verlag, 2003.

[47] M. Kaufmann and J. S. Moore. ACL2. Available at http://www.cs.utexas.edu/users/moore/acl2, November 2007.

[48] H. Kikuchi. Rabin Tree and its Application to Group Key Distribution. In *Automated Technology for Verification and Analysis*, volume 3299 of *Lecture Notes in Computer Science*, pages 384–391. Springer-Verlag, November 2004.

[49] Y. Kim, A. Perrig, and G. Tsudik. Tree-based Group Key Agreement. *ACM Transactions on Information and Systems Security*, 7(1):60–96, 2004.

[50] T. Kropf. *Introduction to Formal Hardware Verification*. Springer-Verlag, 1999.

[51] W. Ku and S. Chen. An Improved Key Management Scheme for Large Dynamic Groups using One-way Function Trees. In *International Conference on Parallel Processing Workshops*, pages 391–396. IEEE Computer Society Press, October 2003.

[52] 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.

[53] M. Layouni, J. Hooman, and S. Tahar. Formal Specification and Verification of the Intrusion-Tolerant Enclaves Protocol. *International Journal of Network Security*, 5(3):288–298, 2007.

[54] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, March 1996.

[55] M. Manulis. Security-Focused Survey on Group Key Exchange Protocols. Technical Report 2006/03, Rohr-University of Bochum, November 2006.

[56] Mastercard and VISA. SET secure electronic transaction specification. Available at http://www.cl.cam.ac.uk/research/security/resources/set/intro.html, May 1997.

[57] L. Mazaré. Computationally Sound Analysis of Protocols using Bilinear Pairings. In *Preliminary Proceedings of International Workshop on Issues in the Theory of Security*, pages 6–21, Braga, Portugal, March 2007.

[58] C. Meadows. Open Issues in Formal Methods for Cryptographic Protocol Analysis. *IEEE Journal on Selected Areas in Communications*, 21(1):44–54, January 2003.

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

[60] 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. *IOS Journal of Computer Security*, 12(6):893–932, 2004.

[61] C. Metayer, J. Abrial, and L. Voisin. RODIN Deliverable 3.2: Event-B Language. Technical Report Project IST-511599, School of Computing Science, University of Newcastle, UK, 2005.

[62] J. Millen and V. Shmatikov. Symbolic Protocol Analysis with an Abelian Group Operator or Diffie-Hellman Exponentiation. *Journal of Computer Security*, 13(3):515–564, 2005.

[63] M. Moharrum, R. Mukkamala, and M. Eltoweissy. A Novel Collusion-Resilient Architecture for Secure Group Communication in Wireless ad-hoc Networks. *Journal of High Speed Networks*, 15(1):73–92, 2006.

[64] J. Nam, J. Paik, U. M. Kim, and D. Won. Resource-aware Protocols for Authenticated Group Key Exchange in Integrated Wired and Wireless Networks. *International Journal on Information Sciences*, 177(23):5441–5467, 2007.

[65] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12), dec 1978.

[66] 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.

[67] L. Paulson. *Isabelle: A Generic Theroem Prover*, volume 828. Springer-Verlag, 1994.

[68] L. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *IOS Journal of Computer Security*, 6(1-2):85–128, 1998.

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

[70] O. Pereira and J. Quisquater. On the Impossibility of Building Secure Cliques-Type Authenticated Group Key Agreement Protocols. *Journal of Computer Security*, 14(2):197–246, 2006.

[71] B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic Security of Reactive Systems. *Elsevier Electronic Notes in Theoretical Computer Science*, 32(4), 2000.

[72] PVS. PVS: A Prototype Verification System, SRI International. Available at http://pvs.csl.sri.com/, July 2008.

[73] S. Rafaeli and D. Hutchison. Hydra: A Decentralised Group Key Management. In *IEEE International Workshops on Enabling Technologies*, pages 62–67. IEEE Computer Society Press, 2002.

[74] S. Rafaeli and D. Hutchison. A Survey of Key Management for Secure Group Communication. *ACM Computing Surveys*, 35(3):309–329, 2003.

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

[76] N. Shankar, S. Owre, J. Rushby, and D. Stringer-Calvert. PVS Prover Guide Version 2.4. Technical report, SRI International, Ca, USA, November 2001.

[77] A. Sherman and D. McGrew. Key Establishment in Large Dynamic Groups Using One-Way Function Trees. *IEEE Transactions on Software Engineering*, 29(5):444–458, 2003.

[78] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman Key Distribution Extended to Group Communication. In *Conference on Computer and Communications Security*, pages 31–37. ACM Press, 1996.

[79] M. Steiner, G. Tsudik, and M. Waidner. CLIQUES: A New Approach to Group Key Agreement. In *International Conference on Distributed Computing Systems*, pages 380–387. IEEE Computer Society Press, 1998.

[80] N. Stouls and M. Potet. Security Policy Enforcement Through Refinement Process. In *Formal Specification and Development in B*, volume 4355 of *Lecture Notes in Computer Science*, pages 216–231. Springer-Verlag, 2007.

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

[82] P. Syverson and C. Meadows. Formal Requirements for Key Distribution Protocols. In *Workshop on the Theory and Applications of Cryptographic Techniques*, volume 950 of *Lecture Notes in Computer Science*, pages 320–331. Springer-Verlag, 1995.

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

[84] K. Verma and J. Goubault-Larrecq. Alternating Two-way AC-tree Automata. *Information Computing*, 205(6):817–869, 2007.

[85] C. Wong, M. Gouda, and S. Lam. Secure Group Communications using Key Graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, 2000.

[86] X. Xu, L. Wang, A. Youssef1, and B. Zhu. Preventing Collusion Attacks on the One-Way Function Tree (OFT) Scheme. In *Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 177–193. Springer-Verlag, June 2007.

[87] J. Zhou. Fixing A Security Flaw in IKE Protocols. *IEEE Electronics Letters*, 35(13):1072–1073, 1999.

Biography

Education

- **Concordia University:** Montreal, Quebec, Canada

  Ph.D candidate, in Electrical Engineering, (Jan. 2003 - present)

- **Concordia University:** Montreal, Quebec, Canada

  M.Eng., in Electrical Engineering, (Sept. 2000 - Dec. 2002)

- **Jordan University of Science and Technology:** Irbid, Jordan

  B.Sc., in Electrical and Computer Engineering, (Sept. 1993 - July 1998)

Work History

- **Hardware Verification Group (HVG):** Concordia University

  Research Assistant, (Sept. 2000 - present)

- **Electrical and Computer Engineering Department:** Concordia University

  Teaching Assistant, (Sept. 2002 - May 2008)

- **Jordan University of Science and Technology:** Department of Computer Engineering, Irbid, Jordan

  Network Administrator, (July 1999 - Aug. 2000)

- **Al-Ahlyya Amman University:** Faculty of Engineering, Amman, Jordan

  Laboratory Engineer, (Oct. 1998 - July 1999)

Publications

- ### Journal Papers

Bio-Jr-1 A. Gawanmeh, A. Bouhoula, and S. Tahar: "Rank Functions based Inference System for Group Key Management Protocols Verification"; International Journal of Network Security, 8(2): 207-218, Science Publications, March 2009.

Bio-Jr-2 A. Gawanmeh, S. Tahar, and K. Winter: "Formal Verification of ASMs using MDGs; Journal of Systems Architecture", 54(1-2): 15-34, Elsevier B.V. Pub., January-February, 2008.

Bio-Jr-3 A. Gawanmeh, S. Tahar, H. Moinudeen, and A. Habibi: "A Design for Verification Approach using an Embedding of PSL in AsmL"; Journal of Circuits, Systems, and Computers, 16(6): 859 - 881, World Scientific Publishing, December 2007.

- **Refereed Conference Papers**

Bio-CF-1 A. Gawanmeh, L. J. Ben Ayed., and S. Tahar: "Event-B based Invariant Checking of Secrecy in Group Key Protocols"; In Proc. IEEE Local Computer Networks Workshop on Network Security. Montreal, Quebec, Canada. IEEE Computer Society Press. (*To appear in October 2008.*).

Bio-CF-2 A. Gawanmeh: "Theorem Proving Based Framework for Verification of Group Key Protocols". In. Proc. International Conference on Theorem Proving in Higher-Order Logics: Emerging Trends Proceedings. Concordia University, Montreal, Quebec, Canada, pp. 9-20, August 2008.

Bio-CF-3 A. Gawanmeh and S. Tahar: "Rank Theorems for Forward Secrecy in Group Key Management Protocols". In. Proc. IEEE Symposium on Frontiers in Networking with Applications. Niagara Falls, Canada, May 2007, pp. 18-23. IEEE Computer Society Press.

Bio-CF-4 A. Gawanmeh, A. Habibi, and S. Tahar: "Embedding and Verification of PSL using ASM"; In Proc. IEEE International Workshop on System on Chip, Cairo, Egypt, December 2006, pp. 125-130. IEEE Computer Society Press.

Bio-CF-5 A. Gawanmeh, A. Habibi, and S. Tahar: "Embedding and Verification of PSL using ASM"; In Proc. International Conference on Abstract State Machines, Paris, France, March 2005, pp. 201-215.

Bio-CF-6 A. Habibi, A. Gawanmeh and S. Tahar: "Assertion Based Verification of PSL for SystemC Designs"; In Proc. IEEE International Symposium on System-on-Chip, Tampere, Finland, November 2004, pp. 177-180. IEEE Computer Society Press.

Bio-CF-7 A. Gawanmeh, A. Habibi and S. Tahar: "Enabling SystemC Verification using Abstract State Machines"; In Proc. Languages for Formal Specification and Verification, Forum on Specification & Design Languages, Lille, France, September 2004.

Bio-CF-8 A. Gawanmeh, S. Tahar, and K. Winter: "Formal Verification of ASM Designs using the MDG Tool"; In Proc. IEEE International Conference on Software Engineering and Formal Methods, Brisbane, Australia, September 2003, pp. 210-219, IEEE Computer Society Press.

Bio-CF-9 A. Gawanmeh, S. Tahar and K. Winter: "Interfacing ASMs with the MDG Tool"; In Proc. Abstract State Machines - Advances in Theory and Applications, Taormina, Italy, March 2003, Lecture Notes in Computer Science vol. 2589, pp. 278-292, Springer-Verlag.

- **Non-Refereed Papers**

Bio-NRCf-1 A. Gawanmeh, S. Tahar, and K. Winter; "A Tool for Verifying ASM

Models using Multiway Decision Graphs"; In Proc. 2003 Micronet Annual Workshop, Toronto, Canada, October 2003, pp. 127-128.

- **Technical Reports**

Bio-Tr-1 A. Gawanmeh, L. Jemni Ben Ayed and Sofiene Tahar: "Verification of Forward Secrecy for Group Key Protocols in Event-B Refinement". Technical Report, Concordia University, Department of Electrical and Computer Engineering, August 2008. [20 pages]

Bio-Tr-2 A. Gawanmeh, L. Jemni Ben Ayed and Sofiene Tahar: "Event-B based Invariant Checking of Secrecy in Group Key Protocols". Technical Report, Concordia University, Department of Electrical and Computer Engineering, May 2008. [20 pages]

Bio-Tr-3 A. Gawanmeh, A. Bouhoula and Sofiene Tahar: "Rank Functions based Inference System for Group Key Management Protocols Verification". Technical Report, Concordia University, Department of Electrical and Computer Engineering, March 2007. [26 pages]

Bio-Tr-4 A. Gawanmeh and Sofiene Tahar: "Rank Theorems for Forward Secrecy in Group Key Management Protocols". Technical Report, Concordia University, Department of Electrical and Computer Engineering, December 2006. [23 pages]

Bio-Tr-5 A. Gawanmeh and Sofiene Tahar: "Formal Specification Requirements for Group Key Management Protocols". Technical Report, Concordia University, Department of Electrical and Computer Engineering, January 2006. [19 pages]

Bio-Tr-6 A. Gawanmeh, A. Habibi, and S. Tahar: "Enabling SystemC Verification using Abstract State Machines"; Technical Report, Concordia University, Department of Electrical and Computer Engineering, May 2004. [15 pages]

Bio-Tr-7 A. Gawanmeh, A. Habibi, and S. Tahar: "An Executable Operational Semantics for SystemC using Abstract State Machines"; Technical Report, Concordia University, Department of Electrical and Computer Engineering, March 2004. [24 pages]

Bio-Tr-8 A. Gawanmeh, S. Tahar and Kirsten Winter: "Formal Verification of ASM Designs using the MDG Tool"; Technical Report, Concordia University, Department of Electrical and Computer Engineering, June 2003. [28 pages]

- **Theses**

Bio-TH-1 Amjad Gawanmeh: "Interfacing Abstract State Machines with Miltiway Decision Graphs". MaSc Thesis, Concordia University, Department of Electrical and Computer Engineering, April 2003.