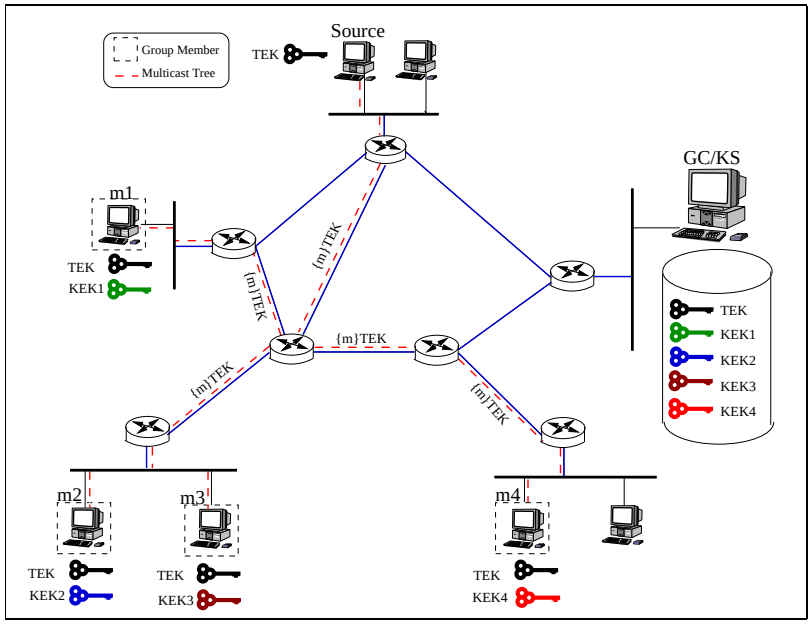
Group Communication Security

Yacine Challal

by Yacine Challal

Group Communication Security

Ph.D. Thesis



Thesis defense: May, 13th 2005

Committee Members

SERGE FDIDA	Université Paris VI	**President**
Professor	Lip6	**Reviewer**
J. WILLIAM ATWOOD	Concordia University	
Professor	Canada	**Reviewer**
PROSPER CHEMOUIL	France Télécom	
Research Director	R&D	
AHMED SERHROUCHNI	ENST	
Assistant Professor	Paris	
HATEM BETTAHAR	Université de Technologie	
Assistant Professor	Compiègne	
ABDELMADJID BOUABDALLAH	Université de Technologie	**Thesis**
Professor	Compiègne	**Advisor**

Dedication

To my parents, and my fiancee
for their patience and support.

Abstract

THE advantages of IP multicast in multi-party communications, such as saving bandwidth, simplicity and efficiency, are very interesting for new services combining voice, video and text over Internet. This urges the effective large scale deployment of multicasting to satisfy the increasing demand for multicasting from both Internet Service Providers (ISPs) and Content Distributors. Unfortunately, the strengths of IP multicast are also its security weaknesses. Indeed, the open and anonymous membership and the distributed nature of multicasting are serious threats to the security of this communication model. Much effort has been conducted to address the many issues relating to securing multicast data transmission, such as: *access control, confidentiality, authentication and watermarking*.

In this thesis we deal with the two keystone security issues of any secure multicast architecture: *data origin authentication and confidentiality*. For each theme, we present a detailed analysis of the problem while highlighting special features and issues inherent to the multicast nature. Then, we review existing solutions in the literature and analyze their advantages and shortcomings. Finally, we provide our own original proposals, depicting their advantages over the previous solutions.

Acknowledgments

MY first and most heartfelt thanks go to my supervisor, M. Abdelmadjid Bouabdallah, Professor at Compiegne University of Technology for invaluable discussions and advice throughout the course of this research, and for many helpful comments and concise and constructive criticism without which this dissertation would not have been achieved.

Particular thanks go to M. Hatem Bettahar, Assistant Professor at Compiegne University of Technology, for inestimable advice and fruitful discussions during my research. Each time I faced difficulties, I had always found Hatem next to me for encouragements and solutions.

I would like also to thank the reviewers of my thesis, M. J. William Atwood, Professor at Concordia University and M. Serge Fdida, Professor at Paris VI University, who took time out of their busy schedules to send valuable comments and feedback about this work. My thanks go also, to M. Prosper Chemouil, Research Director at France Telecom R&D, and M. Ahmed Serhrouchni, Assistant Professor at ENST-Paris, for having accepted to be in the jury of my thesis defense.

Next, a note of gratitude to my colleagues of the Networking group (Heudiasyc Lab.), Imed Romdhani, Mounir Kellil, Hamida Seba, Hani Ragab, Yoann Hinard, Yacine Khaled, Hamid Menouar, David Savourey, Linh Doan, François Clautiaux and Farid Sayeh for their help and encouragement.

I would like to thank our colleagues at Algiers University of Technology (USTHB-Algeria), Professor Nadjib Badache and Mahfoud Benchaiba, and the folks at the Research Centre on Technical and Scientific Information (CERIST-Algeria), Djamel Tandjaoui, Djamel Djenouri, Wahid Derhab for many stimulating discussions of ideas of mutual interest during their visits to our lab.

I wish to thank Said Gharout and Abdelaziz Babakhouya; master students at Bejaia University for the many interesting exchanges during their internship.

I am grateful to the Algerian and French governments for supporting my research with the Algerian-French Cooperation Scholarship.

Special thanks go to Samir Kohil, Ali Khouas, Kamel Merdes, Youcef and the folks at ETM-Ibnrochd for their support and encouragements.

My deep gratitude to my parents, and my family, for supporting me all this time, and to my fiancee, for her support and patience.

Author Publications List

International Journal Publications

1. **Y. Challal**, A. Bouabdallah, Y. Hinard; *"RLH: Receiver driven Layered Hash-chaining for multicast data origin authentication"*, *Computer Communications Journal (Elsevier)*, Volume 28 Issue 7: 726-740, May. 2005.

2. **Y. Challal**, A. Bouabdallah, H. Bettahar; *"H_2A: Hybrid Hash-chaining scheme for Adaptive multicast source authentication of media-streaming"*, *Computers and Security Journal (Elsevier)*, Volume 24 Issue 1: 57-68, Feb. 2005.

3. **Y. Challal**, H. Bettahar, A. Bouabdallah; *"A Taxonomy of Multicast Data Origin Authentication: Issues and Solutions"*, *IEEE Communications Surveys and Tutorials*, Volume 6 number 3: 34-57, Oct. 2004.

4. **Y. Challal**, H. Bettahar, A. Bouabdallah; *"SAKM: A Scalable and Adaptive Key Management Approach for Multicast Communications"*, *ACM SIGCOMM Computer Communications Review*, Volume 34, number 2: 55-70, April 2004.

International Conference Publications

1. H. Ragab Hassan, A. Bouabdallah, H. Bettahar, **Y. Challal**; *"Hi-KD: Hash-based hierarchical Key Distribution for Group Communication"*, poster in *IEEE-INFOCOM'05*, (Miami, FL, USA), March 2005.

2. H. Ragab Hassan, **Y. Challal**, *"Enhanced WEP: a new solution to WEP threats"* in proceedings of *IFIP-Wireless Optical Compter Networks (IFIP-WOCN'05)* (Dubai-UAE), March 2005.

3. **Y. Challal**, A. Bouabdallah, Y. Hinard; *"Efficient multicast source authentication using layered hash-chaining scheme"*, poster in proceedings of *IEEE-Local Computer Networks (IEEE-LCN'04)* (Florida-USA), pages: 411-412, November 2004.

4. **Y. Challal**, H. Bettahar, A. Bouabdallah; *"Hybrid and Adaptive Hash-chaining scheme for data-streaming source authentication"*, in proceedings of *High Speed Networks and Multimedia Communications (HSNMC'04)* (Toulouse-France), Lecture Notes in Computer Science, Volume(3079), pages: 1056-1067.

5. **Y. Challal**, H. Bettahar, A. Bouabdallah; *"A^2Cast: an Adaptive source Authentication protocol for multiCast streams"*, in proceedings of *IEEE-International Symposium on Computer Communications (IEEE-ISCC'04)* (Alexandria-Egypt), pages: 363-368.

6. Y. Khaled, H. Menouar, **Y. Challal**; *"Reactive and Adaptive Protocol for Inter-vehicle Communications"*, in proceedings of *IEEE-International Conference on information and communication Technologies: from Theory to Applications (IEEE-ICTTA'04)* (Damascus-Syria), pages: 63-66.

7. **Y. Challal**, H. Bettahar, A. Bouabdallah; *"A Scalable and Adaptive Key Management Protocol for Group Communication"*, in proceedings of *Wired/Wireless Internet Communications(WWIC'04)*, Lecture Notes in Computer Science (2957), pages: 260-271, Frankfurt(Oder)-Germany, February 2004.

8. H. Bettahar, A. Bouabdallah, **Y. Challal** ; *"AKMP : an Adaptive Key Management Protocol for secure multicast"*, in proceedings of *IEEE-International Conference on Computer Communication Networks (IEEE-IC3N'02)*, pages: 190-195. Miami, October 2002.

National Conference Publications

1. H.R. Hassan, A. Bouabdallah, H. Bettahar, **Y. Challal**; *"Gestion de Clés dans la communication de groupes hiérarchiques"*, in proceedings of *Sécurité et Architectures Réseaux SAR'05*, Batz sur Mer (France) June 2005.

2. **Y. Challal**, H. Bettahar, A. Bouabdallah; *"A Scalable and Adaptive Key Management Approach for Group Communication"*, in proceedings of *Sécurité et Architectures Réseaux SAR'03*, Nancy (France) July 2003.

3. H. Bettahar, A. Bouabdallah, **Y. Challal** ; *"Multicast sécurisé : une approche adaptative"*, in proceedings of *Sécurité et Architectures Réseaux SAR'02* conference. Marrakeche (Moroco), 2002.

Glossary

1-affects-n phenomenon	a protocol suffers from the *1-affects-n* phenomenon if a single membership change in the group affects all the other group members.
A^2Cast	Adaptive source Authentication protocol for multiCAST streams.
Confidentiality	is the property that information is made not available or disclosed to unauthorized individuals, entities, or processes [128].
Data integrity	is the property that data has not been changed, destroyed, or lost in unauthorized or accidental manner [128].
Data origin authentication	is the corroboration that the source of data received is as claimed [128].
DoS	Denial of Service.
GC	Group Controller.
H^2A	Hybrid Hash-chaining scheme for Adaptive multicast data origin authentication.
KEK	Key Encryption Key : a common name given to a key which is used to encrypt another key. This is commonly used to secure the transmission of the group key.
KS	Key Server.
MAC	Message Authentication Code : is a cryptographic mechanism that can be used to assure data origin authentication and data integrity at the same time.
Non-repudiation with proof of origin	provides the recipient of data with evidence that proves the origin of the data, and thus protects the recipient against an attempt by the originator to falsely deny sending the data [128].
Re-keying	The action of distributing a new group key to the current legitimate group members.
RLH	Receiver driven Layered Hash-chaining for multicast data origin authentication.
SAKM	Scalable and Adaptive group Key Management protocol.
TEK	Traffic Encryption Key : the symmetric group key which is used to encrypt and decrypt data sent to the group.

Contents

List of Tables

List of Figures

Introduction

THE phenomenal growth of the Internet in the last few years and the increase of bandwidth in today's networks have provided both inspiration and motivation for the development of new services, combining voice, video and text "over IP". Although unicast communications have been predominant so far, the demand for multicast communications is increasing both from the Internet Service Providers (ISPs) and from content or media providers and distributors. Indeed, *multicasting* is increasingly used as an efficient communication mechanism for group-oriented applications in the Internet such as video conferencing, interactive group games, video on demand (VoD), TV over Internet, e-learning, software updates, database replication and broadcasting stock quotes. Nevertheless, the lack of security in the multicast communication model obstructs the effective and large scale deployment of such strategic business multi-party applications. This limitation motivated a host of research works that have addressed the many issues relating to securing the multicast, such as *confidentiality, authentication, watermarking and access control*. These issues must be seen in the context of the *security policies* that prevail within the given circumstances. For instance, in a *public* stock quotes broadcasting, while *authentication* is a fundamental requirement, *confidentiality* may not be. In the contrary case, both *authentication* and *confidentiality* are required in *video-conference* applications. *Multicast Security* becomes remarkably complex and difficult to comprehend when we consider other ingredients, such as *mobility* and *fault tolerance*. *Mobility* induces the problem of seamless and safe transfer of *security context*. Moreover, suitable mechanisms have to be developed to ensure *access control* for end-users whose *access point* may change during the multicast session. In addition, since the failure of a single security component may compromise safety and privacy of thousands of customers, *robustness* and *fault tolerance* are fundamental and indispensable requirements. In this thesis, we focus on two keystone components of any secure multicast architecture over wired networks: *confidentiality* and *data origin authentication*.

The distribution of data with commercial value or State top-secret content requires the use of appropriate mechanisms to prevent non-legitimate recipients from having access to the content. Besides, the recipient needs to ascertain the origin of the multicast data he receives and the content provider needs also to provide such assurance to protect himself from the dangerous consequences of being impersonated by a fraudulent third party. Even though a multitude of data origin authentication and confidentiality mechanisms currently exist, these security services remain a challenging problem in terms of scalability, efficiency, and performance.

To ensure *confidentiality*, only the customers authorized for the service would have access to the content for only the duration corresponding to their authorization. A straightforward solution is to encrypt the multicast data by the sender with a *group key* common to all authorized recipients. Therefore, this symmetric encryption should prevent other users from having access to the content. However, when the authorized duration for a recipient expires, it is necessary to change the common *group key* into a new key in order to prevent the leaving customer from having access to the content beyond the limit of his authorized duration. Therefore, the sender has to share the new common *group key* with all legitimate recipients except the leaving one. This phase is called *re-keying*, and should be performed each time a customer *joins* the secure session (to prevent him from having access to old content) or *leaves* the session (to prevent him from having access to future content). Thereby, assuring *group communication confidentiality* is a hard problem to solve in case of *large scale* groups with highly dynamic members due to the difficulty of performing frequent *re-keying* securely while inducing low computation, bandwidth and storage overheads.

In the case of *data origin authentication*, it is not possible to use a *common group key* as a means to calculate *authenticators* to multicast messages. Indeed, this straightforward scheme would guarantee that users outside the group cannot impersonate the content provider since only legitimate recipients are supposed to know the *common group key*. However this scheme would not safeguard from internal impersonation where a legitimate recipient calculates an authenticator for a multicast message, using the *group key*, on behalf of the valid source. Therefore, introducing *asymmetry* in the *authentication information generation process* is required in order to allow recipients to verify authentication information without being able to generate it on behalf of the legitimate sender. The keystone problem with data origin authentication in group communication is how to introduce this *asymmetry* in authentication information while inducing low bandwidth and storage overheads. Using digital signatures may induce high computation overhead, and most media-streaming applications cannot afford to sign each packet because of the real-time transmission requirement. An alternative would be to amortize a single signature over a stream of packets. Unfortunately, since most media-streaming applications use an unreliable transport layer, this amortization is likely to be affected by packet loss. The challenge is then how to introduce redundancy in amortization while inducing low bandwidth overhead.

In conclusion, we notice that there are serious conflicts between multicast and security. The anonymous membership based on a single multicast address that makes the openness and efficiency of multicasting, complicates confidentiality which requires individual and explicit identification of members in order to provide them with the right keys to access the encrypted multicast content. Moreover, large scale groups with highly dynamic members present serious scalability issues for *group key management and distribution*. In what relates to authentication, the multi-party nature of multicasting requires the usage of an efficient asymmetric mechanism to provide data origin authentication. Besides, since most media-streaming applications based on multicasting, rely on best effort channels, those asymmetric authentication mechanisms must tolerate packet loss.

Contributions The overall contribution of this dissertation is in the two cornerstone security areas of any secure multicast architecture: *confidentiality and data origin authentication*; two complex

and hot topics that present potential problems. The main contributions are as follows:

1. We proposed a novel *adaptive* scheme for *scalable* group key management with *dynamic* multicast groups, called **SAKM** [28]. What distinguishes our approach is its *dynamism awareness* that allows to tune key management administrative areas in order to achieve better performance trade-offs.

2. We proposed yet another protocol called **AKMP** in [12]. *AKMP* is also an adaptive scheme, but operates at the routing level. This is a promising solution for secure group communication in *ad hoc* networks where all the group members are highly dynamic and involved in the routing function. Indeed, a research team at INRIA research institute has undertaken *AKMP* and adapted it for secure group communication in the context of *ad hoc networks* [17]. In this thesis, we present only *SAKM* which is more developed and better modeled.

3. We proposed an original *taxonomy* of data origin authentication protocols that features out the main common concepts and techniques used by the proposed solutions. The presentation of the solutions is followed by deep comparisons and discussions that are fruitful for both academic and industrial researchers [25].

4. We proposed an efficient *data origin authentication* protocol for group communication called H_2A [29]. Our protocol trades better bandwidth overhead for tolerance to packet loss, thanks to the new *adaptive hash-chaining* scheme that allows to adapt the authentication information to packet loss variation over time.

5. We proposed another efficient *data origin authentication* protocol for group communication called **RLH** [31]. This new protocol focuses on the problem of the variation of packet loss over space because of the not-uniform nature of packet loss distribution. Simulations show that our protocol trades better bandwidth overhead for tolerance to packet loss, thanks to its new *layered hash-chaining* scheme that allows receivers to adapt the authentication information to actual packet loss ratio faced in their respective subnets.

Outline This thesis is divided into two parts that reflect the two security services we have dealt with in our work: *confidentiality* and *data origin authentication*. The first part deals with group communication confidentiality and focuses particularly on group key management issues. This part is in turn divided into three chapters. In the first chapter we recall some useful definitions and we state the problem of group confidentiality, then we picture out key management requirements from different points of view. In a second chapter, we review existing key management solutions in the literature through an original taxonomy that allows to grasp better the concepts underlying the proposed schemes. Finally, the third chapter is dedicated to our proposal: the *Scalable and Adaptive Key Management* protocol which uses a novel *adaptive* scheme that achieves better performance trade-offs. The second part focuses on data origin authentication in group communication with lossy channels. Similarly, this part will be further subdivided into four chapters. In the first one, we recall some definitions and depict the requirements of data origin authentication

from different points of view. Then we present a taxonomy of existing solutions with comparisons and discussions. The two last chapters are dedicated to our proposals: *Source-driven and Receiver-driven Adaptive hash-chaining schemes* that deal with trading bandwidth for tolerance to packet loss to assure data origin authentication and non-repudiation in lossy channel networks. We end-up this work with some concluding remarks and we highlight the main future work directions and open issues in multicast security.

Chapter 1

Multicast Security Background

WE focus, in this thesis, on two fundamental aspects of multicast security: *confidentiality* and *data origin authentication* for multi-party applications. In this chapter we introduce and motivate the work in this thesis. First, we recall multicast strengths. Then, we present security threats in the multicast model and the IRTF/IETF framework of multicast security in order to give an overview of the components of a secure multicast architecture.

1.1 Multicasting: strengths and motivations

Multicasting is an efficient communication mechanism for group-oriented applications such as video-conferencing, interactive group games, video on demand (VoD), TV over Internet, e-learning, database replication and broadcasting stock quotes. IP multicast [40] saves bandwidth by sending the source traffic on a multicast tree that spans all the members of the group. In this communication model, groups are identified by a group address and any node of the network can join or leave the group freely (using the Internet Group Management Protocol (IGMP)[37, 53, 21] with IPv4 or Multicast Listener Discovery (MLD)[39, 132] for IPv6). Figure 1.1 illustrates the basic IP multicast components.

The multicast source, sends multicast data to a specific multicast address. IGMP is running between the subnet-routers and the attached hosts. Each subnet-router sends (periodically) a *IGMP-Query* to ask hosts on its subnet whether they are interested in some multicast sessions. Any host interested in the current session sends a *IGMP-Report* to the subnet multicast router, indicating the address of the session. Upon receiving this join request, the subnet-router runs with other routers a multicast routing protocol (such as: DVMRP[133], MOSPF[90], PIM-DM[1], PIM-SM[38, 50], CBT[8, 6]) that allows to graft the new member to the multicast spanning tree. When a host leaves the session, its multicast subnet-router prunes him from the multicast tree, if no more hosts in the attached segments are still members in the session.

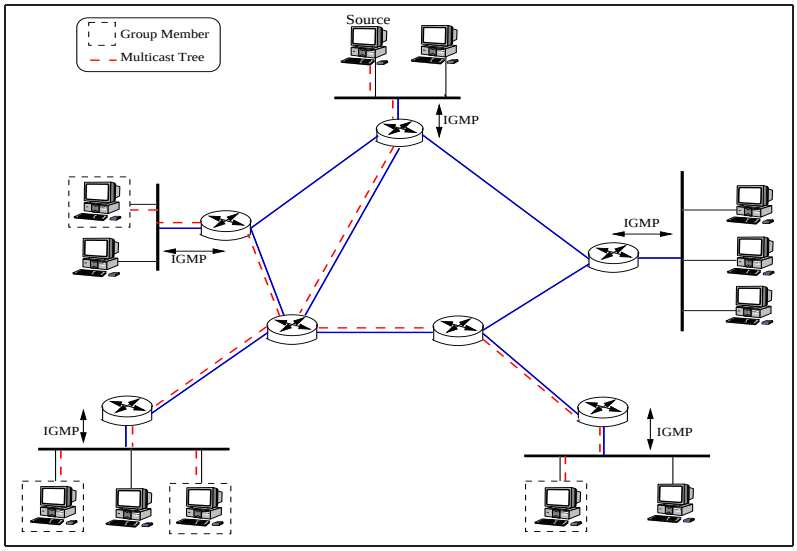


Figure 1.1: Basic components of IP Multicast

1.2 Security and Multicasting: a Complex Deal

The IP multicast model is attractive because it can scale to a large number of members. However, scalability is achieved due to the fact that no host identification information is maintained by the routers [64]. Any host in a subnet can join a multicast group without its subnet router passing identification information about the host to other routers in the distribution tree. This simplicity which makes the strength of multicast routing, presents however, many vulnerabilities [7, 69]:

1. IP multicast does not support closed groups. In fact, multicast addresses are publicly known: joining or leaving a group does not require specific permissions. Hence, any user can join a multicast group and receive messages sent to the group.

2. There is no access control to a multicast group: an intruder can send data to the group without being a valid member, and disturbs the multicast session or eventually create bottlenecks in the network (Denial of Service attack).

3. Data sent to the group may transit via many unsecure channels. Thus, eavesdropping opportunities are more important.

1.2.1 Security threats and countermeasures

There exist many security threats, inherent to the distributed and open nature of IP multicast, that require security countermeasures (cf. figure 1.2).

1. *Denial of Service / Access Control*: in the basic IP multicast model, any node can send data to a multicast session, and any node can become a member of any multicast session. It is

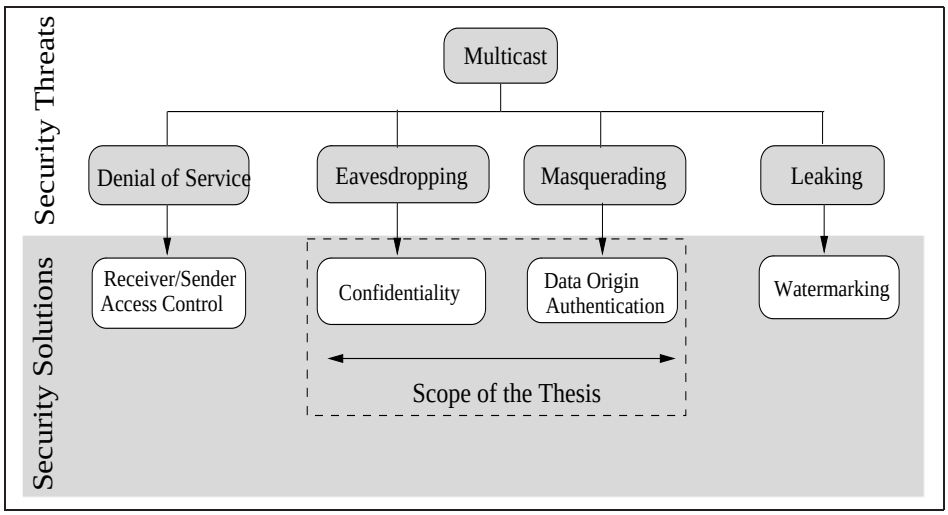


Figure 1.2: Multicast Security Threats and their Countermeasures

clear that this model is vulnerable to Denial of Service (DoS) attacks, where fraudulent users join or send data to multicast sessions only to waste bandwidth or to overwhelm other group members with garbage data or malicious code. Solving these problems requires controlling the ability of hosts to send data or to join a multicast tree distribution. These are called respectively: *sender and receiver access control*. Kellil et al. surveyed the main proposed schemes in [72].

2. *Eavesdropping / Confidentiality*: in unicast communication, two users can provide confidentiality by encrypting data with a shared key. In multicast communication, a group key is given to every authorized member. This group key is used by the sender as a symmetric key to encrypt the multicast traffic. This becomes complicated when group membership is dynamic (members join and leave continuously the multicast session). Research work in *group key management* aims to provide efficient re-keying schemes for dynamic membership groups. Many authors surveyed independently the existing solutions, such as Kruus in [77], Hardjono and Tsudik in [64], Judge and Ammar in [69], Rafaeli and Hutchison in [114], and Seba et al. in [124].

3. *Masquerading / Data Origin Authentication*: data origin authentication is the ability of group members to verify the identity of the sender of a received packet. There has been work that aims to efficiently provide this level of authentication. We surveyed proposed approaches and protocols in [25].

4. *Leaking / Watermarking*: encryption is generally used to safeguard content while it is being transmitted so that unauthorized persons cannot read the stream from the network, but this offers no protection after the intended receiver receives the data. There is no protection against unauthorized duplication and propagation by the intended receiver. Watermarking can provide protection in the form of theft deterrence. Watermarking is the embedding of some identifying information into the content in such a way that it cannot be removed by the

user but can be extracted or read by the appropriate party [69]. Judge and Ammar surveyed some existing solutions in [69].

In this thesis we focus only on *group key management* and *data origin authentication* in multi-party communications.

1.2.2 Challenges to overcome

In order to illustrate the difficulty of securing group communications, let us consider the scenario depicted in figure 1.1. Suppose that the source multicasts confidential content with a commercial value, and receivers subscribe and pay to receive this data during different intervals of time. The multicast source faces two major security issues to distribute its content to its customers over the Internet:

1. *Confidentiality:* only the customers who paid for the service would have access to the content for only the duration corresponding to the payment. A straightforward solution is to encrypt the multicast data by the source with a key K common to all recipients who paid for the service. Therefore, this symmetric encryption should prevent other users from having access to the content. However, when the authorized duration for a recipient R_i expires, it is necessary to change K into a new key K' in order to prevent R_i from having access to the content beyond the limit of his payment. Therefore, the source has to share the new key K' with all legitimate recipients except R_i. This phase is called *re-keying*. The challenging problem to assure *re-keying* is how to distribute the new key K' securely while inducing a low computation overhead, and how to distribute it efficiently while inducing low bandwidth overhead (number or required messages) and storage overhead (storage of required intermediate keys). These challenges become stronger with large scale groups with highly dynamic members and make *group communication confidentiality* a difficult problem to apprehend.

2. *Data origin Authentication:* the source requires to guard itself from the disastrous consequences of being impersonated by another entity who might generate content on its behalf that harms its credibility. On the other hand, costumers require to be ascertained that the received content originates effectively from the claimed source. A straightforward solution is again to use a common key K to compute a Message Authentication Code (MAC) of the sent messages. Recipients use the same key for verification. This technique guarantees that users outside the group cannot impersonate the source since only legitimate recipients are supposed to know the key K. However this technique does not safeguard from internal impersonation where a legitimate recipient calculates a MAC using K on behalf of the source. The challenge with data origin authentication in group communication is how to introduce asymmetry in authentication information while inducing low bandwidth and storage overheads. Indeed, asymmetry is required in order to allow recipients to verify the authentication information without being able to generate it on behalf of the legitimate sender. Using digital

signatures may induce high computation overhead, and most of media-streaming applications cannot afford signing each packet because of real time transmission requirement. An alternative would be to amortize a single signature over a stream of packets. Unfortunately, since most of media-streaming applications use unreliable transport layer, this amortization is likely to be affected by packet loss, and hence not allowing to verify authentication of the entire stream. The challenge is then how to introduce redundancy in amortization while inducing low bandwidth overhead.

We notice that there are serious conflicts between multicast and security. Indeed, the anonymous membership based on a single multicast address that makes the scalability of multicasting, complicates confidentiality which requires individual and explicit identification of members in order to provide them with the right keys to access the encrypted multicast content. In what relates to authentication, the multi-party nature of multicasting requires the usage of an efficient asymmetric mechanism to provide data origin authentication. Besides, since most of media-streaming applications based on multicasting, rely on a best effort channels, those asymmetric authentication mechanisms must tolerate packet loss.

1.3 The IETF multicast security reference framework

Multicast security motivated intensive academic and industrial research works. Therefore, dedicated research groups are created at the IETF and IRTF to address the many issues relating to standardizing secure multicast. These research groups: *GSEC (IRTF)* and *MSEC (IETF)* defined a framework to standardize multicast security efforts. The framework defines a set of functional building blocks that should be tackled for any secure multicast architecture with centralized or distributed designs [61, 63, 65] (cf. figure 1.3).

The reference diagram contains boxes and arrows. The boxes are the functional entities and the arrows are the interfaces between them. Standard protocols are needed for the interfaces, which support the multicast services between the functional entities. There are three sets of functional entities in both centralized and distributed designs:

1. *Group Controller Key Server*: the Group Controller Key Server (GCKS) represents both the entity and functions relating to the issuance and management of cryptographic keys used by a multicast group, and functions pertaining to group membership management such as authentication and authorization of candidate members. In a distributed architecture, several GCKS entities are involved in the key management related services in order to achieve scalability .

2. *Sender and Receiver*: the sender is an entity that sends data to the multicast group. In a *1-to-n* multicast group, only a single sender is allowed to transmit data to the group. In an *m-to-n* multicast group, many (or even all) group members can transmit data to the group. Both

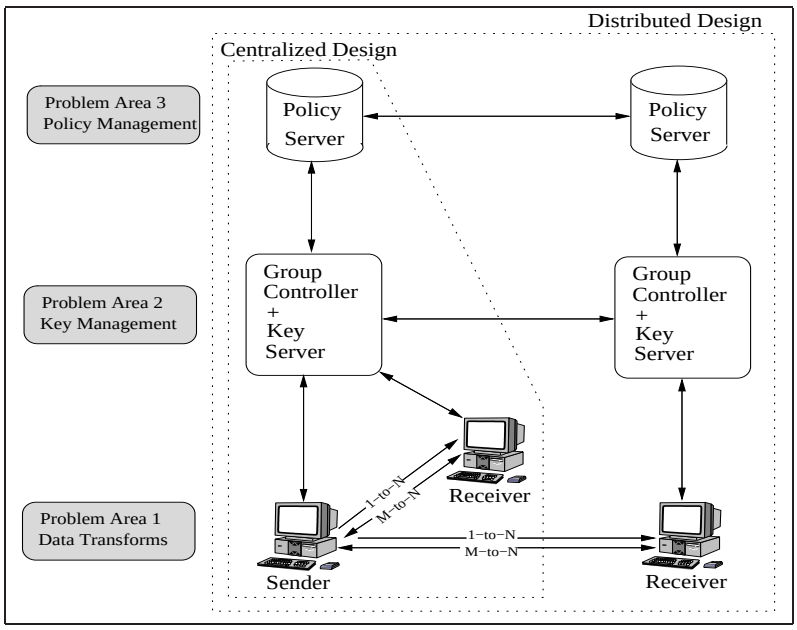


Figure 1.3: The IETF multicast security reference framework

sender and receiver must interact with the GCKS entity for the purpose of key management. This includes user authentication, the obtaining of keying material in accordance with key management policies for the group, obtaining new keys during key updates, and obtaining other messages relating to the management of keying material and security parameters.

3. *Policy server*: the policy server represents both the entity and functions used to create and manage security policies specific to a multicast group. The policy server interacts with the GCKS in order to install and manage the security policies related to the membership of a given multicast group. The interactions between the policy server and other entities in the reference framework are dependent to a large extent on the security circumstances being addressed by a given policy.

In figure 1.3 we distinguish three main functional areas relating to multicast security:

1. *Data handling area*: This area covers problems concerning the security related treatments of multicast data by the sender and the receiver, such as encryption and authentication. In particular, algorithms for efficient application of the cryptographic keys in the multicast context need to be studied.

2. *Key management area*: The management of the cryptographic keys of a group requires the management of their associated state and parameters. Solutions for specific problems must be addressed. These may include the following:

   - Methods for member identification and authentication;
   - Methods to verify the membership to groups;

- Methods to establish a secure channel between a GCKS entity and a member for the purpose of key distribution;
- Methods of re-keying in case of membership change;
- etc.

The needs related to key management must be seen in the context of the policies that prevail within the given circumstances.

3. *Multicast Security Policy area*: Multicast security policies must provide the rules for operation of the other elements of the Reference Framework. Multicast security policy management should extend the concepts developed for unicast communication and policy representation.

1.4 Conclusion

In this chapter we showed the advantages of using multicasting in multi-party communications, such as saving bandwidth, simplicity and efficiency. These advantages are typically interesting for new services combining voice, video and text over Internet. Therefore the demand for multicasting is increasing from both Internet Service Providers (ISPs) and Content Distributors. Unfortunately, the strengths of multicast make its security weaknesses. Indeed, the open and anonymous membership and the distributed nature of multicasting are serious threats to the security of this communication model. Many efforts have been conducted to define and build countermeasures to the multicast security threats such as: *access control, confidentiality, authentication and watermarking*. The IETF/IRTF dedicated special groups (MSec and GSec) to tackle security threats in multicasting. They defined a general framework in order to feature standard functional building blocks and protocols for multicast security. In the following chapters, we will focus on two cornerstone security issues of any secure multicast architecture: *confidentiality* and *data origin authentication*.

Part I

Group Communication Confidentiality

Chapter 2

Definitions and Requirements

IN this chapter we recall some cryptographic mechanisms to ensure data confidentiality. Then we introduce the problem statement of confidentiality in the context of group communication. Finally, we summarize the requirements of confidentiality in group communication from different points of view.

2.1 Data confidentiality

Data confidentiality is the property that information is made not available or disclosed to unauthorized individuals, entities, or processes [128]. *Confidentiality* is guaranteed using *encryption* which is a cryptographic transformation of data (called *plain text*) into a form (called *cipher text*) that conceals the data's original meaning to prevent it from being known or used. If the transformation is reversible, the corresponding reversal process is called *decryption*, which is a transformation that restores encrypted data to its original state [128].

With most advanced cryptography, the ability to keep encrypted information secret is based not on the cryptographic encryption algorithm, which is widely known, but on a piece of information called a *key* that must be used with the algorithm to produce an encrypted result or to decrypt previously encrypted information. Depending on whether the same or different keys are used to encrypt and to decrypt the information, we distinguish between two types of encryption systems:

2.1.1 Symmetric-key Encryption

In a symmetric-key encryption system, a *secret key* is shared between the sender and the receiver and it is used by the sender to encrypt the message before its transmission and by the receiver to decrypt the received message. The encryption of the message produces a non-intelligible piece of information and the decryption reproduces the original message. Examples of symmetric-key encryption systems are: DES[109], AES[111], IDEA[71]. Table 2.1 gives some measurements for usually used symmetric encryption algorithms. These measurements are from a benchmark due to Dai [35]. The author used Pentium 4 2.1 GHz processor under Windows XP SP 1.

Encryption Algorithm	Encryption Speed (MBytes / second)
DES	22.19
IDEA	17.65
AES-128 bits	62.04

Table 2.1: Computation speed of some encryption algorithms

2.1.2 Public-key Encryption

Public-key encryption (also called asymmetric encryption) involves a pair of keys (a *public key* and a *private key*) associated with the sender. The *public key* is published, and the corresponding *private key* is kept secret by the sender. Data encrypted with the sender's *public key* can be decrypted only with the sender's *private key*. Before its transmission, the message is encrypted with the receiver's public key, and the receiver of the encrypted data decrypts it with the corresponding private key. Compared with symmetric-key encryption, public-key encryption induces high computation overhead and is therefore not appropriate for large amounts of data. However, public-key encryption is generally used to send a secret symmetric key, which is then used to encrypt additional data. An example of asymmetric encryption systems is: RSA [117].

2.2 Group Communication Confidentiality

In this section, we will use a simple scenario to introduce the challenging issues relating to group confidentiality and key management. We consider a *source* that sends data to a set of *receivers* in a multicast session. The security of the session is managed by two main *functional entities*: a *Group Controller (GC)* responsible for authentication, authorization and access control, and a *Key Server (KS)* responsible for the maintenance and distribution of the required *key material*. Note that these two functions can be implemented over a single physical entity or over different physical entities depending on the key management architecture. Figure 2.1 depicts this simple scenario.

To ensure confidentiality during the multicast session, the sender (source) shares a secret symmetric key with all valid group members, called *Traffic Encryption Key (TEK)*. To multicast a secret message, the source encrypts the message with the TEK using a symmetric encryption algorithm. Upon receiving the encrypted multicast message $\{m\}_{TEK}$, each valid member that knows the TEK can decrypt it with TEK and recover the original one. To avoid that a leaving or an ejected member from the group, continues to decrypt the secret multicast messages, the KS must generate a new TEK and securely distribute it to the all remaining group members except the leaving one. This operation is called ***re-keying***. The KS shares a secret key called *Key Encryption Key* (KEK_i) with each member m_i (cf. figure 2.1). To re-key the group following a leave from the group, the KS generates a new TEK: TEK', and sends it to each member m_i (except the leaving one) encrypted with its corresponding KEK_i. Thereby, the leaving member cannot know the new TEK' and hence will not be able to decrypt future multicast messages of this session.

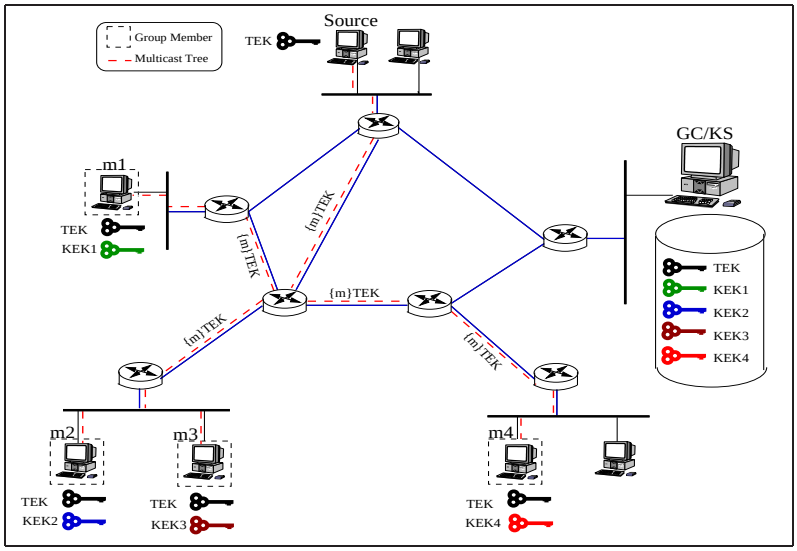


Figure 2.1: Simple scenario of group confidentiality components

When a new member joins the session, it must be authenticated by the GC. After checking the rights of the new member to access the group, the KS proceeds to a new group *re-keying* to avoid that the new member decrypts previous exchanged messages using the current key. Therefore, the KS generates a new TEK: TEK', encrypts it with the old TEK: $\{TEK'\}_{TEK}$, and multicasts it to the group. Hence all old members can recover the new TEK: TEK'. Then, the KS encrypts TEK' with the secret KEK_j that it shares with the new member m_j and sends it to him to recover TEK' which is required to decrypt the multicast messages.

The maintenance and the distribution of the keys involved in *re-keying* and encryption is commonly called : *Group Key Management*. In this illustrative protocol, re-keying induces a $O(n)$ re-key messages after each *leave* from the group, where n is the number of group members. It induces also a storage of $O(n)$ keys (1 *TEK* + n KEK_i) at the KS during the whole secure multicast session. Since each membership change in the group requires re-keying and the group may be highly dynamic, one of the challenges of *group key management* is how to assure re-keying using the minimum bandwidth overhead without increasing the storage overhead. Proposed solutions in the literature, as we will see in the following chapter, trade bandwidth overhead for storage overhead to achieve the best overall performance.

2.3 Group Key Management Requirements

Efficient group key management protocols should take into consideration miscellaneous requirements. Figure 2.2 summarizes these requirements from four points of view: security, quality of service, KS's resources, and group members' resources.

Security requirements:

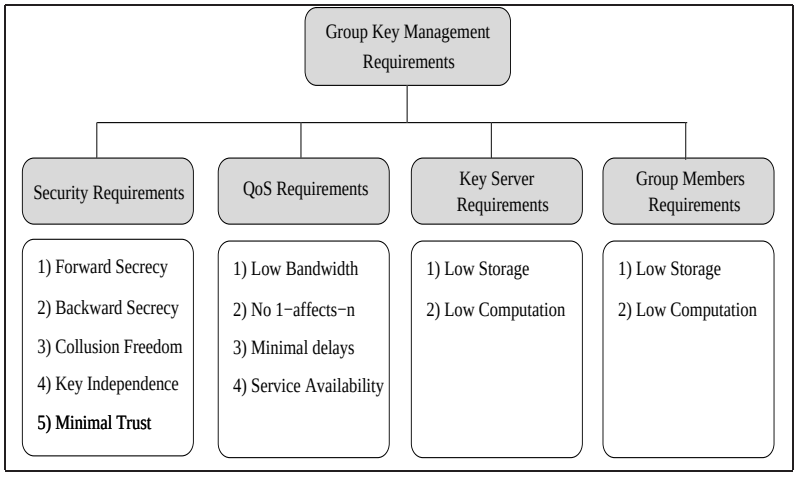


Figure 2.2: Group Key Management Requirements

1. *Forward secrecy* requires that users who left the group should not have access to any future key. This ensures that a member cannot decrypt data after it leaves the group. To assure forward secrecy, a re-key of the group with a new TEK after each leave from the group is the ultimate solution.

2. *Backward secrecy* requires that a new user that joins the session should not have access to any old key. This ensures that a member cannot decrypt data sent before it joins the group. To assure backward secrecy, a re-key of the group with a new TEK after each join to the group is the ultimate solution.

3. *Collusion freedom* requires that any set of fraudulent users should not be able to deduce the current traffic encryption key.

4. *Key independence*: a protocol is said *key independent* if a disclosure of a key does not compromise other keys.

5. *Minimal trust*: the key management scheme should not place trust in a high number of entities. Otherwise, the effective deployment of the scheme would not be easy.

Quality of service requirement:

1. *Low bandwidth overhead*: the re-key of the group should not induce a *high number of messages*, especially for dynamic groups. Ideally, this should be independent from the group size.

2. *1-affects-n*: a protocol suffers from the *1-affects-n* phenomenon if a single membership change in the group affects all the other group members. This happens typically when a single membership change requires that all group members commit to a new TEK.

3. *Minimal delays*: many applications that are built over the multicast service (typically, multi-media applications) are sensitive to jitters and delays in packet delivery. Therefore, any key

management scheme should take this into consideration and hence minimizes the impact of key management on the *delays of packet delivery*.

4. *Service availability*: the failure of a single entity in the key management architecture must not prevent the operation of the whole multicast session.

Other requirements:

1. The key management scheme must not induce neither high *storage of keys* nor high *computation overhead* at the key server or group members.

2.4 Conclusion

Group communication confidentiality requires that only valid users could decrypt the multicast data even if the data is broadcast to the entire network. Therefore, a symmetric key must be shared securely between the source and valid group members. This key is used to encrypt data by the source and to decrypt it by valid members to recover original data. This key is generally called: *Traffic Encryption Key (TEK)*. To assure backward and forward secrecy a re-key process must be triggered after each membership change (join or leave) in the group. It consists in generating a new TEK and distributing it to the members including the new one in case of a join or to the residual members in case of a leave. A critical problem with any re-keying technique is *scalability*: since the re-keying process should be triggered after each membership change, the number of TEK update messages may be important in case of highly dynamic groups, and thereby induces high bandwidth overhead and what is commonly called *1-affects-n* phenomenon [87].

In the next chapter, we will review different solutions proposed in the literature that deal with key management to assure confidentiality for group communications.

Chapter 3

A Taxonomy of Group Key Management

GROUP key management is an important functional building block for any secure multicast architecture. Thereby, it has been extensively studied in the literature. In this chapter we present relevant group key management protocols. Then, we compare them against some pertinent performance criteria.

As we notice from the previous chapters, a critical problem with any re-key technique is scalability: as a re-keying process should be triggered after each membership change, the number of *TEK* update messages may be important in case of frequent join and leave operations. Thereby, some solutions propose to organize the secure group into subgroups with independent local *TEKs*. This reduces the impact of re-keying, but requires data transformation at the borders of subgroups as we will see in the following sections. Therefore, we can classify existing solutions into two approaches: the *Common TEK* approach and the *TEK per sub-group approach* as illustrated in figure 3.1. In what follows, we present each class of protocols and we further refine the classification in order to highlight the underlying common concepts and mechanisms. We will illustrate each identified sub-category with relevant protocols from the literature.

3.1 Common TEK Approach

In this approach, all group members share a *common* Traffic Encryption Key (TEK). The management of this single key can be further classified into three classes: *centralized*, *decentralized* or *distributed*. Figure 3.1 illustrates this classification.

3.1.1 Centralized Protocols

In this approach, the key distribution function is assured by a single entity which is responsible for generating and distributing the traffic encryption key (TEK) whenever required. In figure 3.1, centralized protocols are further classified into three sub-categories depending on the technique used to distribute the TEKs. In what follows, we present each sub-category:

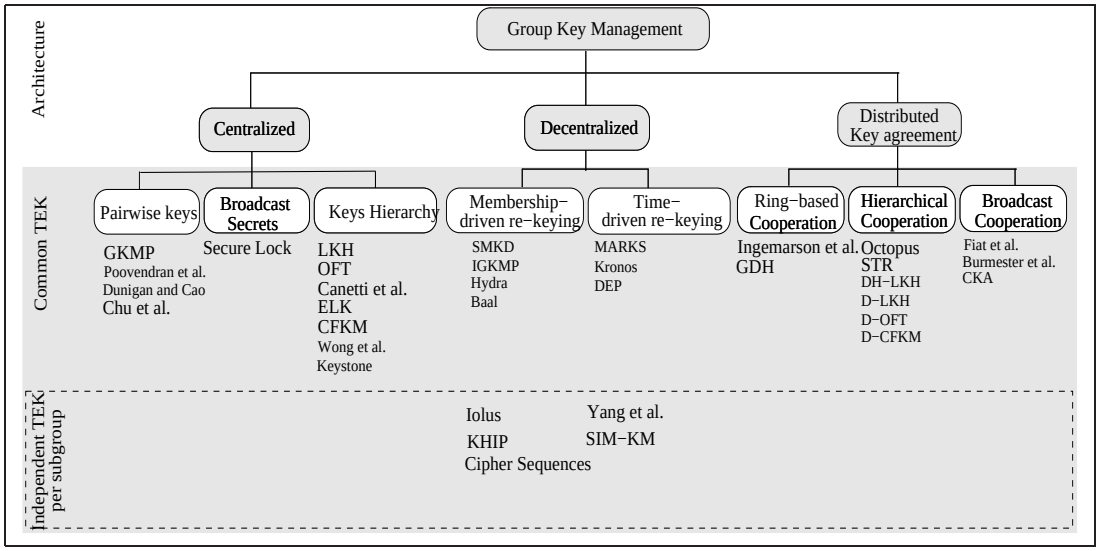


Figure 3.1: Taxonomy of *Common TEK* Group Key Management Protocols

Pairwise Keys

In this sub-category of protocols, the *Key Server* shares a secret key with each group member. These keys are generally called: *Key Encryption Keys (KEK)* and are used to establish secure channels between the KS and each member in order to re-distribute the current TEK securely whenever required.

GROUP KEY MANAGEMENT PROTOCOL (GKMP): Harney and Muckenhirn [66, 67] proposed the Group Key Management Protocol (GKMP) that uses this approach. The key server shares a secret key with each valid group member (*KEKs*). In GKMP, the key server generates a Group Key Packet (GKP) that contains two keys: a Group TEK (GTEK) and a Group KEK (GKEK). The GTEK is used to encrypt the traffic and the GKEK is used to secure the distribution of a new GKP whenever required. When a new member joins the session, the key server generates a new GKP (which contains a new GTEK to assure backward secrecy) and sends it securely to the new member encrypted with the KEK established with this new member, and sends it to the other members encrypted with the old GTEK. The key server refreshes the GKP periodically and uses the GKEK for its distribution to the group members. When a member leaves the group, the key server generates a new GKP and sends it to each remaining member encrypted with the KEK that it shares with each member. Thus to assure forward secrecy, GKMP requires $O(n)$ re-key messages for each leave from the group. Therefore, this solution does not scale to large groups with highly dynamic members.

Dunigan and Cao [47] proposed a similar protocol that suffers from the same issues. Poovendram et al. [108] have also proposed a similar scheme to GKMP, where authentication and authorization functions are delegated to other group members rather than centralized at the same group controller entity.

HAO-HUA CHU ET AL. PROTOCOL: In the solution proposed by Hao-hua Chu et al. in [34], a *Group Leader* shares a secret *Key Encryption Key (KEK)* with each group member. To send a secret multicast message m, the sender encrypts m with a random key k. Then, the sender encrypts k with the secret KEK that it shares with the group leader, and sends it to the group along with the encrypted message. Upon receiving the message, receivers cannot decrypt it since they do not know the random key k. When the leader receives the message, it decrypts k using the key that it shares with the source and constructs a validation message which contains k encrypted with each KEK that the leader shares with a valid group member (excluding the departing members). Upon receiving the validation message, each receiver decrypts k using its KEK and hence decrypts m which was encrypted using k. This protocol has the drawback to require the transmission of the *validation* multicast message by the group leader, with a size in the order of $O(n)$ (n being the number of current valid group members), after each time the source sends a message to the group.

Broadcast Secrets Approach

In this sub-category of protocols, the re-keying of the group is based on broadcast messages instead of peer to peer secret transmissions.

SECURE LOCKS: Chiou and Chen [33] proposed Secure Lock; a key management protocol where the key server requires only a single broadcast to establish the group key or to re-key the entire group in case of a leave. The protocol relies on the following theorem:

Theorem 1 *Chinese Remainder Theorem Let $m_1, \ldots, m_n$ be pairwise relatively prime positive integers, and let $a_1, \ldots, a_n$ be any integers. Then the system of linear congruences in one variable given by:*

$$x \equiv a_1 \ mod \ m_1$$

$$\ldots$$

$$x \equiv a_n \ mod \ m_n$$

has a unique solution modulo $M = m_1 \times m_2 \times \ldots m_n$.

The unique solution is: $x = \sum_{i=1}^{n} a_i M_i y_i \bmod M$, where $M_i = M/m_i$ and $y_i = M_i^{-1} \bmod m_i$.

In this protocol, the key server assigns a positive integer m_i to each member and shares a secret value k_i with each of them. When the server wants to send a message to the group, it generates a random value K and uses it to encrypt the message. Then, it encrypts K with each secret k_i and obtains the set $\{K_i\}$ of the encryptions of K ($K_i = \{K\}_{k_i}$). Then the server computes a lock M which is the solution to the equation system:

$$M \equiv K_1 \bmod m_1$$

$$\ldots$$

$$M \equiv K_n \bmod m_n$$

Then the server multicasts the lock M as well as the encrypted message with K. Upon reception of the lock M, each member recovers the encryption key $K = \{M \bmod m_i\}_{k_i}$, and hence decrypts the received message. Only members whose secret k_i and its corresponding positive integer m_i are included in the computation of the lock M, can recover the decryption key K.

This protocol minimizes the number of re-key messages. However, it increases the computation at the server due to the Chinese Remainder calculations before sending each message to the group.

Hierarchy of Keys Approach

We showed that in the *pairwise keys* approach, re-keying induces a high number of update messages (in the order of $O(n)$, with n being the number of group members). This is due mainly to the fact that the key server establishes a secure channel individually with each member and uses this channel to distribute the TEK updates. In order to reduce the number of update messages, in this sub-category of protocols, the key server shares secret keys with sub-groups of the entire secure group in addition to the individual channels. Then, when a member leaves the secure session, the key server uses the secret sub-group keys, that are unknown by the leaving member, to distribute the new TEK. Thereby, sub-group secret keys allow to reduce the required number of update messages. In what follows we present some protocols that use this concept for re-keying:

LOCAL KEY HIERARCHY (LKH): Independently, Wong et al. [136, 137] and Wallner et al. [135] proposed the Logical Key Hierarchy (LKH) protocol. In LKH, the key server maintains a tree of keys. The nodes of the tree correspond to KEKs and the leaves of the tree correspond to secret keys shared with the members of the group. Each member holds a copy of its leaf secret key and all the KEKs corresponding to the nodes in the path from its leaf to the root. The key corresponding to the root of the tree is the TEK. For a balanced binary tree, each member stores at most $1 + log_2(n)$ keys, where n is the number of group members.

This *key hierarchy* allows to reduce the number of re-key messages to $O(\log n)$ instead of $O(n)$ in GKMP.

Example: Let us consider a multicast group with six members $\{u_1, u_2, u_3, u_4, u_5, u_6\}$. The key server builds a hierarchy of keys as shown in figure 3.2. Each member owns a secret key which is a leaf in the tree as well as all the keys on its path to the root. The root represents the TEK shared by the members. The other keys are used to reduce the required re-keying messages. According to figure 3.2 : u_1 owns{ k_1, k_{12}, k_{1234}, TEK}, u_2 owns{ k_2, k_{12}, k_{1234}, TEK}, u_3 owns{ k_3, k_{34}, k_{1234}, TEK}, u_4 owns{ k_4, k_{34}, k_{1234}, TEK}, u_5 owns{ k_5, k_{56}, TEK} and u_6 owns{ k_6, k_{56}, TEK}.

Let us assume that u_5 leaves the group, KS updates k_{56} into k'_{56}, sends k'_{56} to u_6 encrypted with k_6. TEK is updated into TEK' and sent to $\{u_1, u_2, u_3, u_4\}$ encrypted with k_{1234} and to u_6 encrypted with k'_{56} and hence only three messages are required instead of five messages if GKMP were used.

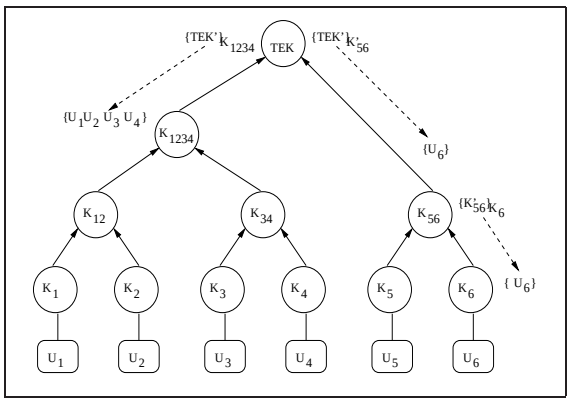


Figure 3.2: key Hierarchy

Wong et al. [136, 137] proposed the extension of the binary key tree to a k-ary key tree. Using a greater degree reduces the number of keys maintained by the members because of a smaller tree depth. Performance analysis shows that optimal results are reached with trees having a degree less or equal to 4. The authors propose also the Keystone architecture [140], where the key server is aided by secondary controllers called registrars, whose role is limited to registration and authentication of new members.

ONE-WAY FUNCTION TREES (OFT): McGrew and Sherman [82, 4] proposed an improvement over LKH called One-way Function Trees (OFT). OFT allows to reduce the number of re-key messages from $2\log_2(n)$ to only $\log_2(n)$. With OFT, a KEK is calculated by members rather than attributed by the key server. Indeed, each KEK k_i is computed using its child KEKs using the formula:

$$k_i = f(g(k_{left(i)}), g(k_{right(i)})) \tag{3.1}$$

where $left(i)$ and $right(i)$ denote respectively the left and right children of node i, and g is a one-way function. The result of applying g to a key k: $g(k)$, is called the *blinded key* version of k.

In this protocol, each member maintains its leaf secret key and its *blinded sibling* key and the set of blinded sibling KEKs of its *ancestors*. Figure 3.3 illustrates the ancestors and their corresponding sibling keys of member $u2$.

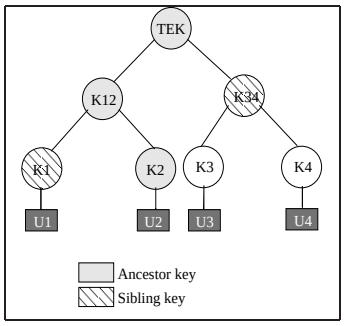


Figure 3.3: Ancestor and Sibling keys of member U_2.

Using formula 3.1, each member can calculate all the required ancestor KEKs (KEKs on the nodes in the path from the leaf secret to the root) recursively. In the original scheme (LKH), when a new KEK is generated, it is encrypted with its two child KEKs. However, in OFT when a blinded key is changed in a node it has to be encrypted only with the key of its sibling node. Hence, the required number of re-key messages is reduced by half.

Example: Let us consider the hierarchy of keys in figure 3.4.

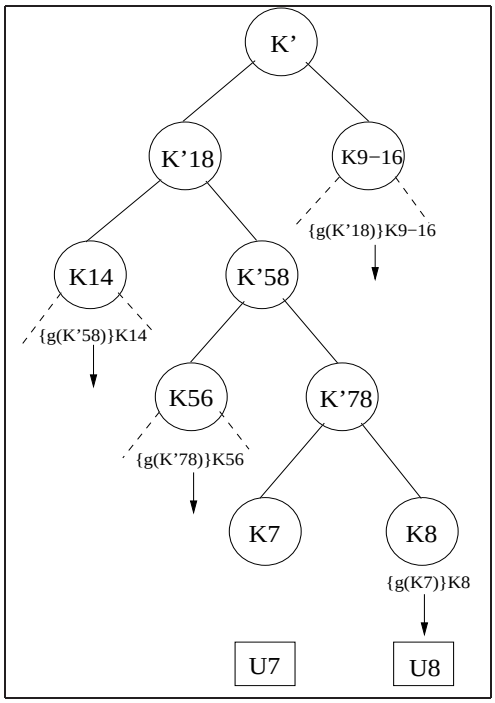


Figure 3.4: Example of a *OFT* scenario. U_7 joins the session.

When user U_7 joins the group, the keys K_{78}, K_{58}, K_{18} and the group key K, should be modified into K'_{78}, K'_{58}, K'_{18} and K', respectively. In order to redistribute the new group key and the modified *KEK*s, the only values that should be sent, are the *blinded* keys: $g(K_7)$, $g(K'_{78})$, $g(K'_{58})$, and $g(K'_{18})$, respectively encrypted with: K_8, K_{56}, K_{14}, and K_{9-16}. The new *TEK* and *KEK*s can be now calculated as follows: $K'_{78} = f(g(K_7), g(K_8))$, $K'_{58} = f(g(K_{56}), g(K'_{78}))$, $K'_{18} = f(g(K_{14}), g(K'_{58}))$, and $K' = f(g(K'_{18}), g(K_{9-16}))$. In this example, user U_8 maintains K_8, $g(K_{56})$, $g(K_{14})$, and $g(K_{9-16})$. When it receives $\{g(K_7)\}_{K_8}$, it calculates recursively all the keys on its path to the root of the hierarchy, using the above formulas. These calculations culminate into the new group key K'.

Canetti et al. [22] proposed a similar approach that has the same communication overhead. The proposed scheme called: one-way function chain tree, uses a pseudo-random-generator to generate the new KEKs rather than a one-way function. Perrig et al. proposed yet another similar protocol called: Efficient Large group Key distribution (ELK) [106], that uses Pseudo Random Functions to generate the new KEKs.

CENTRALIZED FLAT TABLE KEY MANAGEMENT (CFKM): Waldvogel et al. [134] proposed the Centralized Flat Table Key Management protocol (CFKM). In this approach, the key hierarchy is replaced by a flat table in order to reduce the number of keys maintained by the *Key Server*. The table contains a single entry for the TEK and $2w$ entries for the KEKs, where w is the number of bits in a member identifier (the authors propose to use IP addresses as member identifiers). Two keys are associated to the two possible values of each bit in a member id. Figure 3.5 illustrates the structure of the table for $w = 4$.

TEK	
KEK00	KEK01
KEK10	KEK11
KEK20	KEK21
KEK30	KEK31

Figure 3.5: CFKM table with $w = 4$

Each member holds the KEKs associated to the values of its identifier bits. Thus, each member holds $w + 1$ keys (w KEKs in addition to TEK). For instance, a member with the identifier 0101 maintains KEK00, KEK11, KEK20, KEK31 and the TEK. When a member leaves the group, all the keys held by this departing member should be modified to assure forward secrecy. Therefore the key server sends a re-key message containing two parts: a first part contains the TEK encrypted with each not compromised KEK from the flat table, and hence all the remaining members would be able to decrypt the new TEK. The second part contains the new KEKs encrypted with both the old KEK and the new TEK. This way, the leaving member cannot recover the new TEK and the remaining members can update their old KEKs without having access to the KEKs of other members. Figure 3.6 illustrates the re-key message sent by the key server after the leave of the member with the identifier 0101.

	TEK	
id bit # 0	{{KEK00new}KEK00old}TEKnew	{TEKnew}KEK01
id bit # 1	{TEKnew}KEK10	{{KEK11new}KEK11old}TEKnew
id bit # 2	{{KEK20new}KEK20old}TEKnew	{TEKnew}KEK21
id bit # 3	{TEKnew}KEK30	{{KEK31new}KEK31old}TEKnew
	bit 0	bit 1

Figure 3.6: CFKM re-key message when member 0101 leaves the group

Comparison

In table 3.1 we compare the above protocols against the following relevant criteria:

Protocol	1-affects-n	Re-key Overhead			Storage Overhead	
		Join		Leave	Key Server	Member
		Multicast	Unicast			
GKMP	Yes	2	2	$2n$	$n+2$	3
LKH	Yes	$\log_2(n)-1$	$\log_2(n)+1$	$2\log_2(n)$	$2n-1$	$\log_2(n)+1$
OFT	Yes	$\log_2(n)+1$	$\log_2(n)+1$	$\log_2(n)+1$	$2n-1$	$\log_2(n)+1$
CFKM	Yes	$2I$	$I+1$	$2I$	$2I+1$	$I+1$
Secure Lock	No	0	2	0	2n	2

n: number of group members, I: number of bits in a member id.

Table 3.1: Comparison of centralized group key management

1. *1-affects-n*: a protocol suffers from the *1-affects-n* phenomenon if a single membership change in the group affects all the other group members.
2. *Storage at the key server*: the number of keys that should be maintained by a key server.
3. *Storage at a member*: the number of keys that should be maintained by a group member.
4. *Join re-key overhead*: number of re-key messages sent by the key server to redistribute the group TEK after a join.
5. *Leave re-key overhead*: number of re-key messages sent by the key server to redistribute the group TEK after a leave.

The GKMP protocol achieves an excellent result for storage at the members. However, this result is achieved by providing no method for re-keying the group after a member has left, except re-creating the entire group which induces a $O(n)$ re-key messages overhead, with n being the number of the remaining group members. Secure Lock achieves also excellent results for storage and communication overheads on both members and the key server. However, these results are achieved by increasing the computation overhead at the key server due to the Chinese Remainder calculations. So far, the best solutions for centralized group key management appear to be those using a hierarchical tree of KEKs. They achieve good overall results without compromising any aspects of security.

3.1.2 Decentralized Architectures

In this approach, a hierarchy of key managers share the labor of distributing the TEK to group members in order to avoid bottlenecks and single point of failure. We distinguish two sub-categories corresponding to the case where the TEK is modified after each membership change (*membership driven*), or systematically after each slot of time (*time driven*) (cf. figure 3.1).

Membership-driven re-keying

In this sub-category of protocols, the TEK is changed each time a join or a leave occurs in the membership of the group. In what follows we present some protocols relying on this approach.

SCALABLE MULTICAST KEY DISTRIBUTION (SMKD): Ballardie proposed in RFC1949 [5] the Scalable Multicast Key Distribution (SMKD); a protocol that exploits the tree built by the Core Based Tree multicast routing protocol (CBT) [8, 6] to deliver keys to multicast group members. In a CBT architecture, the multicast tree is rooted at a main core. Secondary cores can exist eventually. The main core creates an access control list (ACL), a session key GTEK and key encryption key GKEK used to update the session key GTEK. The ACL, the GTEK and the GKEK are transmitted to secondary cores and other nodes when they join the multicast tree after their authentication. Any router or secondary core authenticated with the primary core can authenticate joining members and use the ACL to distribute the keys, but only the main core generates those keys. With SMKD there is no solution for forward secrecy other than to recreate an entirely new group without the leaving members.

INTRA-DOMAIN GROUP KEY MANAGEMENT PROTOCOL (IGKMP): DeCleene et al. [62, 36] proposed the Intra-domain Group Key Management Protocol IGKMP. This architecture divides the network into administratively scoped areas. There is a Domain Key Distributor (DKD) and many Area Key Distributors (AKD). Each AKD is responsible for one area. Figure 3.7 illustrates an example of this architecture.

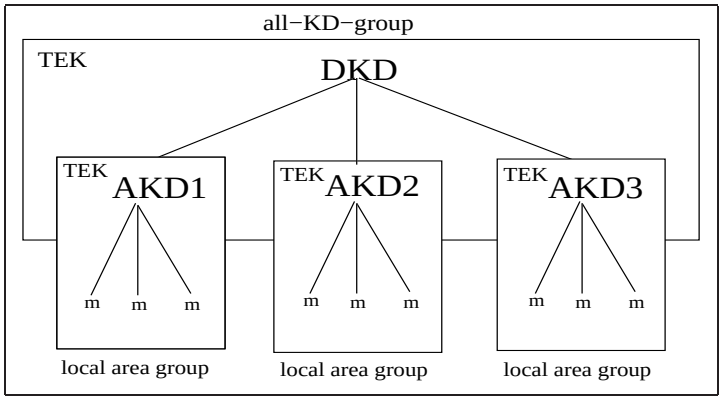


Figure 3.7: An example of an Inter-domain Group Key Management (IGKMP) Architecture

The group TEK is generated by the DKD and is propagated to the group members through the AKDs. The DKD and AKDs belong to a multicast group called All-KD-Group. The DKD uses this group to transmit re-key messages to the AKDs who re-key in turn their respective areas. This architecture suffers from a single point of failure which is the DKD; the sole entity responsible for generating the group TEK. Besides, in case of an AKD failure, members belonging to the same area will be not able to access the group communication.

HYDRA: Rafeli and Hutchison [113] proposed Hydra protocol, in which the group is organized into smaller sub-groups, and a server called the Hydra server (HS_i) controls each sub-group i. If a membership change occurs at sub-group i, the corresponding HS_i generates the group TEK and sends it to the other HS_js involved in that session. In order to have the same group TEK distributed to all HSs a special protocol is used to ensure that only a single valid HS is generating the new TEK whenever required.

BAAL Chaddoud et al. [23] proposed a similar protocol called Baal which defines three entities:

1. *The group controller (GC)*: maintains a participant list (PL) and creates and distributes the group key (TEK) to group members via local controllers.
2. *Local controllers (CL)*: the GC delegates a LC to each subnet (generally a local network) to manage the keys within its subnet. When a LC receives a new TEK it distributes it to the members connected to its subnet. Besides, a LC can play the role of the GC by generating and distributing new TEKs after membership changes following some coordination rules.
3. *Group member*: a member in the PL.

When a membership change occurs at a subnet, the corresponding LC can generate a new TEK and distribute it to its subnet and to the other members via their LCs. To assure that a single LC generates a new TEK at a time, the GC assigns a priority to each LC and when many LCs distribute simultaneously a new TEK, the LCs are instructed to commit to the TEK issued by the LC having the highest priority.

R. Oppliger and A. Albanese [95] proposed a similar decentralized solution called *Distributed Registration and Key distribution (DiRK)*. In their architecture, the authors distinguish between *active* and *passive* group members. *Active* members have the ability to register other new members, and to generate and distribute *TEK*s whenever required. *DiRK* relies on a *PKI* to authenticate *active* members, and the origin of received *TEK*s.

Time-driven re-keying

In this sub-category of protocols, the TEK is changed after each specific period of time. Thereby, the departing members are not excluded immediately from having access to the secure content. Similarly, new members are appointed to wait for the beginning of a new interval of time before having access to the content. In what follows we present some protocols relying on this concept.

KRONOS: Setia et al. [125] proposed the Kronos protocol. This protocol is driven by periodic re-keying rather than membership changes, which means that a new group TEK is generated after each period of time rather than after each membership change. Similarly to IGKMP, in Kronos a domain is divided into areas managed by different AKDs. However, in Kronos the DKD does not multicast the group TEK each time to the AKDs. Instead of that, each AKD generates independently the same group TEK whenever required and re-keys the members belonging to its area. To implement this scheme, the AKDs' clocks should be synchronized, and the AKDs have to agree on a re-key period. Second, the DKD transmits secret factors K and R_0 to AKDs using secure channels. To generate the group TEK R_{i+1}, AKDs calculate after each period of time: $R_{i+1} = E_K(R_i)$, which is the encryption of the previous TEK (R_i) with the encryption algorithm E using the secret key K.

MARKS: In MARKS, Briscoe [19] suggests slicing the time length to be protected (such as the transmission time of a TV show) into small portions of time and using a different key for encrypting each slice. The encryption keys are leaves in a binary hash tree that is generated from a single seed. A blinding function, such as MD5 [116] is used to create the tree nodes. Figure 3.8 shows an example of the generated binary tree whose leaves are the keys that correspond to the different slices.

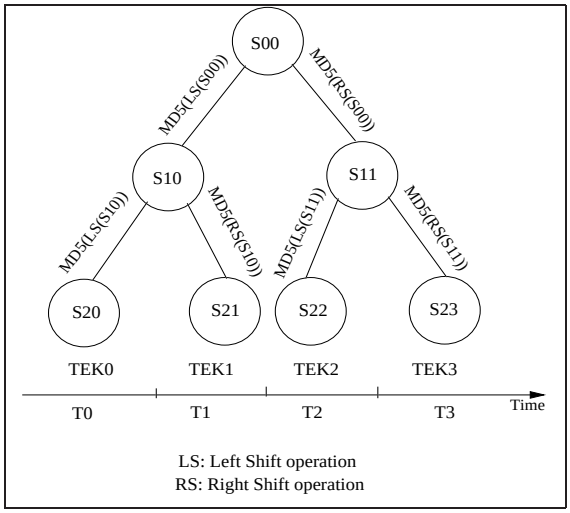


Figure 3.8: An example of a MARKS key generation tree

Each intermediate node (including the root) allows to generate two children (left and right children). The left node is generated by shifting its parent one bit to the left and applying the blinding function on it. The right node is generated by shifting its parent one bit to the right and applying the blinding function on it (cf. figure 3.8). Users willing to access the group communication receive the seeds needed to generate the required keys. The system cannot be used in situations where a membership change requires the change of the group key, since the keys are changed as a function of the time. The distribution of the seeds and the management of receivers' queries are assured by a set of key managers.

DUAL ENCRYPTION PROTOCOL (DEP): A common drawback of most of decentralized protocols is the involvement of a high number of intermediary parties. In practice it is difficult to assume trustiness for all of these entities. In order to solve the problem of trusting third parties, Dondeti et al. [45, 46, 44] proposed the Dual Encryption Protocol (DEP). In their work, they suggest hierarchical sub-grouping of the group members where a sub-group manager (SGM) controls each sub-group. There define three types of KEKs and one Data Encryption Key (DEK): KEK_{i1} is shared between a SGM_i and its sub-group members. KEK_{i2} is shared between the Key Server (KS) and the group members of sub-group i excluding SGM_i. Finally, KS shares KEK_{i3} with SGM_i. In order to distribute the DEK to the group members, the KS generates and transmits a package containing the DEK encrypted with KEK_{i2} and encrypted again with KEK_{i3}. Upon receiving the package, SGM_i decrypts its part of the message using KEK_{i3} and recovers the DEK

Protocol	Key independence	1-affects-n	Local re-key	Data transformation
SMKD	Yes	Yes	No	No
IGKMP	Yes	Yes	No	No
DEP	Yes	Yes	No	No
Kronos	No	-	No	No
Hydra	Yes	Yes	No	No
MARKS	No	-	No	No

Table 3.2: Comparison of some decentralized group key management protocols

encrypted with its sub-group KEK (KEK_{i2}), which is not known by the SGM_i. SGM_i encrypts this encrypted DEK using KEK_{i1} shared with its sub-group members and sends it out to sub-group i. Each member of sub-group i decrypts the message using KEK_{i1} and then, decrypting the message using KEK_{i2} (shared with KS), recovers DEK. Therefore, the DEK cannot be recovered by any entity that does not know both keys. Hence, although there are third parties involved in the management (SGMs), they do not have access to the group key (DEK). When the membership of sub-group i changes, the SGM_i changes KEK_{i1} and sends it to its members. Future DEK changes cannot be accessed for members of sub-group i that did not received the new KEK_{i1}.

Comparison

In table 3.2, we compare some of the above protocols against the following criteria:

1. *Key independence*: a protocol is said *key independent* if a disclosure of a key does not compromise other keys.
2. *1-affects-n*: a protocol suffers from the *1-affects-n* phenomenon if a single membership change in the group affects all the other group members.
3. *Local re-keying*: membership changes in a sub-group should be treated locally.
4. *Data transformation*: data is transformed using some means when messages pass from a sub-group to another.

Kronos does not provide key independence because it generates new keys based on old ones, and if any past key is compromised , all future keys are disclosed. The same happens with MARKS if a seed is compromised.

We note that even though these protocols divide the whole group into sub-groups they still suffer from *1-affects-n* phenomenon because of using the same TEK for all sub-groups.

3.1.3 Distributed Key-agreement Protocols

With distributed key-agreement protocols, the group members cooperate to establish a group key. This improves the reliability of the overall system and reduces the bottlenecks in the network in

comparison to the centralized approach. In figure 3.1, we further classify the protocols of this category into three sub-categories depending on the virtual topology created by the members for cooperation.

Ring-based cooperation

In this sub-category, the cooperation of group members forms a virtual ring, as we will see in the following protocols.

INGEMARSON ET AL. PROTOCOL: The protocol proposed by Ingemarson et al. in [68] is one of the earliest propositions to extend Diffie-Hellman key agreement protocol [42] to group communication. In this protocol, members are organized into a virtual ring; in a way that member M_i communicates with member M_{i+1} and member M_n with member M_1. The group members compute the group key within $(n-1)$ rounds. Initially, each member M_i generates a random value N_i, computes g^{N_i} and sends it to the next member M_{i+1}. Then, in each round, each member M_i raises to the power N_i the intermediate value received from member M_{i-1}, and sends the result to the member M_{i+1}. Each member performs n exponentiations and gets the group key $K_n = g^{N_1 N_2 \ldots N_n}$ after $(n-1)$ rounds. This protocol is not suitable for dynamic groups because it requires the execution of the entire algorithm after each membership change.

GROUP DIFFIE-HELLMAN (GDH): Steiner et al. [131] proposed this group key exchange protocol as an extension of the Diffie-Hellman protocol [42] to establish group keys. The group agrees on a pair of primes (q and α) and starts calculating in a distributed way the intermediate values. The first member calculates the first value (α^{x_1}, with x_1 a random secret generated by the first member) and sends it to the next member. Each subsequent member receiving the set of intermediary values, raises them using its own secret number generating a new set: a set generated by the i^{th} member will have i intermediate values with $i-1$ exponents and a cardinal value containing all exponents. For example; the fourth member receives the set: $\{\alpha^{x_2 x_3}, \alpha^{x_1 x_3}, \alpha^{x_1 x_2}, \alpha^{x_1 x_2 x_3}\}$ and generates the set $\{\alpha^{x_2 x_3 x_4}, \alpha^{x_1 x_3 x_4}, \alpha^{x_1 x_2 x_4}, \alpha^{x_1 x_2 x_3}, \alpha^{x_1 x_2 x_3 x_4}\}$. The cardinal value in this example is $\alpha^{x_1 x_2 x_3 x_4}$. The last member can easily calculate the group key k from the cardinal value: $k = \alpha^{x_1 x_2 x_3 \ldots x_n} \mod q$. The last member raises all intermediate values to its secret value and multicasts the whole set to all group members. Upon receiving this message, each member j extracts its respective intermediate value and calculates k by exponentiation of the j^{th} value to x_j. The setup time and the length of messages are linear in terms of the number of group members n since all members must contribute to generating the group key.

Hierarchy-based cooperation

OCTOPUS: Becket and Wille [9] proposed the Octopus protocol. This protocol is also based on Diffie-Hellman key exchange protocol [42]. In Octopus, the large group (composed of n members) is split into four sub-groups ($\frac{n}{4}$ members each). Each sub-group agrees internally on an intermediate DH value: $I_{subgroup} = \alpha^{u_1 u_2 \ldots u_n/4}$, where u_i is the contribution from user i, and then the

subgroups exchange their intermediary values. All group members can then calculate the group key. The leader member in each sub-group is responsible for collecting contributions from its sub-group members and calculating the intermediary DH value ($I_{subgroup}$). Let us call the subgroup leaders A, B, C and D. First, A and B, using DH, exchange their intermediary values I_a and I_b, creating $\alpha^{I_a.I_b}$. Also, C and D do the same and create $\alpha^{I_c.I_b}$. Then, A and C exchange $\alpha^{I_a.I_b}$ and $\alpha^{I_c.I_d}$. Leaders, B and D do the same. Now, all of them can calculate $\alpha^{I_a.I_b.I_c.I_d}$. After that, A, B, C and D send to their respective subgroups $\alpha^{\frac{I_a.I_b.I_c.I_d}{u_i}}$, where $i = 1 \ldots (n-4)/4$, and all members of the group are capable of calculating the group key.

SKINNY TREE (STR): The *Skinny TRee (STR)* protocol, proposed by Steer et al. [130] and undertaken by Kim et al. [74], is a contributive protocol using a tree structure. Figure 3.9 illustrates an STR tree with four members.

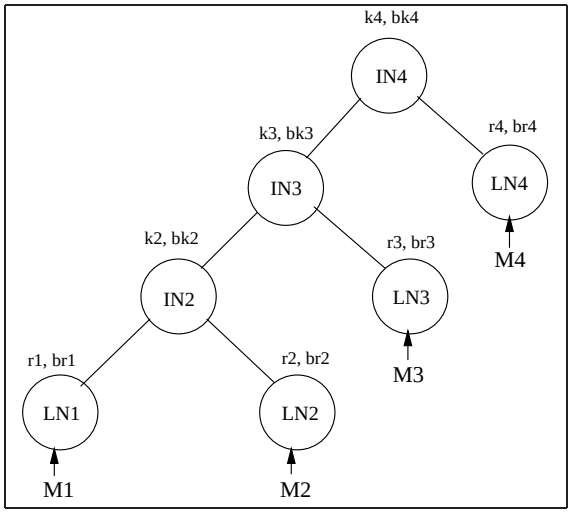


Figure 3.9: An example of a STR tree

The leaves are associated to group members. Each leaf is identified by its position LN_i in the tree and holds a secret random r_i (generated by the corresponding member M_i) and its public blinded version $br_i = g^{r_i} \bmod p$, where g and p are DH parameters. Each internal node is identified by its position IN_i in the tree and holds a secret random k_i and its blinded public version $bk_i = g^{k_i} \bmod p$. Each secret k_i is recursively calculated as follows: $k_i = (bk_{i-1})^{r_i} \bmod p = (br_i)^{k_{i-1}} \bmod p$. The group key is the key associated to the root: $k_n = g^{r_n g^{r_{n-1} \ldots g^{r_2 r_1}}}$. Due to the linear structure of the tree, this solution induces a $O(n)$ key calculations in order to establish the group key associated to the root of the tree. Besides, each member should store and maintain all the public keys associated to all the nodes of the tree. In case of a membership change (join / leave) the tree is re-built consequently and hence all the members update the group key which is the new key k_n associated to the root of the tree.

DIFFIE HELLMAN - LOGICAL KEY HIERARCHY (DH-LKH): Perrig et al. [101, 73] proposed a variant of STR using a binary tree (less deeper) in order to reduce the number of key calculations from the order of $O(n)$ to $O(\log(n))$. Indeed the proposed protocol is a distributed version of

LKH. The tree is built recursively from bottom to up. Initially, each member M_i generates a random r_i as a secret key associated to its leaf. To build upper level of the tree, two members: one as a leader of a left subtree and another one as a leader of a right subtree, broadcast their respective DH computations and hence allow to all the members to calculate the group key corresponding to the root of the tree. Using the tree in figure 3.10, intermediate keys are $k_{12} = \alpha^{k_1 k_2} \bmod p$, $k_{34} = \alpha^{k_3 k_4} \bmod p$ and the group key is $k_{14} = \alpha^{k_{12} k_{34}} \bmod p$.

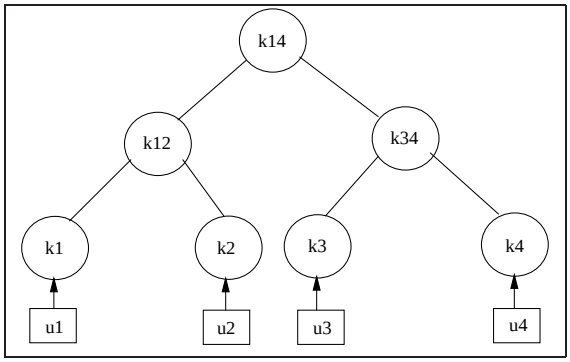


Figure 3.10: LKH Tree

DISTRIBUTED LOGICAL KEY HIERARCHY (D-LKH): A similar distributed approach based on the logical key hierarchy has been proposed by Rodeh et al. in [118]. In this approach, the Key Server is completely abolished and the logical key hierarchy is generated among the members, therefore there is no entity that knows all the keys at the same time. This protocol uses the notion of subtrees agreeing on a mutual key. This means that two groups of members, namely subtree L and subtree R, agree on a mutual encryption key. Member m_l is assumed to be L's leader and member m_r is R's leader. Subtree L has subtree key k_L and subtree R has subtree key k_R. The protocol used to agree on a mutual key goes as follows:

1. Member m_l chooses a new key k_{LR}, and sends it to member m_r using a secure channel.
2. Member m_l encrypts key k_{LR} with key k_L and multicasts it to members of subtree L; member m_r encrypts key k_{LR} with key k_R and multicasts it to members of subtree R.
3. All members $(L \cup R)$ receive the new key.

Similarly, Dondeti et al. [43] and Waldvogel et al. [134] propose distributed versions of OFT [82, 4] (D-OFT) and CFKM [134] (D-CFKM) protocols respectively.

Broadcast-based approach

In this approach, the key agreement relies on broadcasting secret messages and distributed computations that culminate into the group key.

FIAT AND NAOR PROTOCOL: Fiat and Naor [54] proposed a protocol that relies on Diffie-Hellman property that consists in the requirement of each member to broadcast a single message to the other participants in order to agree on a common secret. In the proposed protocol, a reliable center T initializes the system. T choses two primes q_1 and q_2 and broadcasts $p = q_1.q_2$ to all nodes. Then, T generates a random g and keeps it secret. When a new member M_i joins the group, T sends to this new member two values: a random x_i (which is relatively prime with each other x_j previously generated for members M_j), and a key $\alpha_i = g^{x_i} \bmod p$. M_i keeps α_i secret. To agree on a group key K, each member broadcasts its values x_i and hence each of them calculates $K = \alpha_i^{\prod_{j=1,j\neq i}^{n} x_j} \bmod p = g^{x_1 x_2 \ldots x_n} \bmod p$. This protocol is not robust against collusions as shown in [54], and has the drawback to require a reliable third party: T.

BURMESTER AND DESMEDT PROTOCOL: Burmester and Desmedt [20] proposed a very efficient protocol that executes in only three rounds:

1. member m_i generates its random exponent r_i and broadcasts $Z_i = \alpha^{r_i}$;
2. member m_i computes and broadcasts $X_i = (Z_{i+1}/Z_{i-1})^{r_i}$;
3. member m_i can now compute the key $K_n = Z_{i-1}^{nr_i}.X_i^{n-i}.\ldots.X_{n-2} \bmod p$.

The group key calculated by each member is then $K_n = \alpha^{N_1N_2+N_2N_3+\ldots N_nN_1}$. This protocol requires $n+1$ exponentiations per member and in all but one the exponent is at most $n-1$. The drawback is the requirement of $2n$ broadcast messages.

CONFERENCE KEY AGREEMENT (CKA): Boyd [18] proposed yet another protocol for conference key agreement (CKA) where all group members contribute to generating the group key. The group key is generated with a combining function: $K = f(N_1, h(N_2), \ldots, h(N_n))$, where f is the combining function (a MAC), h is a one-way function, n is the group size, and N_i is the contribution from group member i. The protocol specifies that $n-1$ members broadcast their contributions (N_i) in the clear. The group leader, for example U_1, encrypts its contribution (N_1) with the public key of each member and broadcasts it. All group members who had their public key used to encrypt N_1 can decrypt it and generate the group key.

Comparison

In table 3.3 we compare the presented distributed key management protocols against the following criteria:

1. *Number of rounds*: the number of rounds required before the members commit to a group key.
2. *Number of messages*: the number of messages required to establish the group key.
3. *DH exchange*: whether the protocol is based on Diffie-Hellman key exchange.

Scheme	Nb. rounds	Nb. messages		DH exchange	Leader required
		multicast	unicast		
Ingemarson et al.	$n-1$	0	$n(n-1)$	Yes	No
GDH	n	n	$n-1$	Yes	No
Octopus	$2(n-1)/4+2$	0	$3n-4$	Yes	Yes
STR	n	n	0	Yes	No
DH-LKH	$\log_2 n$	$\log_2 n$	0	Yes	No
D-LKH	3	1	n	No	Yes
D-OFT	$\log_2 n$	0	$2\log_2 n$	No	No
D-CFKM	n	0	$2n-1$	No	No
Fiat et al.	2	n	n	Yes	Yes
Burmester et al.	3	$2n$	0	No	No
CKA	3	n	$n-1$	No	Yes

Table 3.3: Comparison of Distributed Key Management Protocols

4. *Leader required*: whether the protocol requires the existence of a leader or leaders for the operation of the key agreement protocol.

The protocols that do not rely on a group leader have an advantage over those with a group leader because, without a leader, all members are treated equally and if one or more members fail to complete the protocol, it will not affect the whole group. In the protocols with a group leader, a leader failure is fatal for creating the group key and the operation has to be restarted from scratch. We did not consider the *1-affects-n* phenomenon because in distributed protocols all the members are contributors in the creation of the group key and hence all of them should commit to the new key whenever a membership change occurs in the group.

3.2 Independent TEK per sub-group

The *common TEK* approach has the drawback to require that all group members commit to a new TEK, whenever a membership change occurs in the group, in order to ensure *perfect backward and forward secrecy*. This is commonly called *1-affects-n* phenomenon. In order to mitigate the *1-affects-n* phenomenon, another approach consists in organizing group members into subgroups. Each sub-group uses its own independent TEK. Indeed, in this scheme when a membership change occurs in a subgroup, it affects only the members belonging to the same sub-group. The existing protocols that use independent TEK per sub-group fall into two sub-categories: the *membership-driven re-keying* protocols that do re-keying after each membership change, and *time-driven re-keying* protocols that do batch re-keying after each period of time. Figure 3.1 illustrates this classification.

3.2.1 Membership-driven re-keying

IOLUS: Mittra [87] proposed Iolus, a framework of a hierarchy of multicast sub-groups. Each sub-group is an independent multicast group (with its own multicast address and eventually its

own multicast routing protocol). The overall sub-groups form a virtual multicast group. Each sub-group is managed by a Group Security Agent (GSA) which is responsible for key management inside the sub-group. A main controller called the Group Security Controller (GSC) manages the GSAs. Figure 3.11 illustrates a hierarchy with six sub-groups. Each of them uses its own TEK.

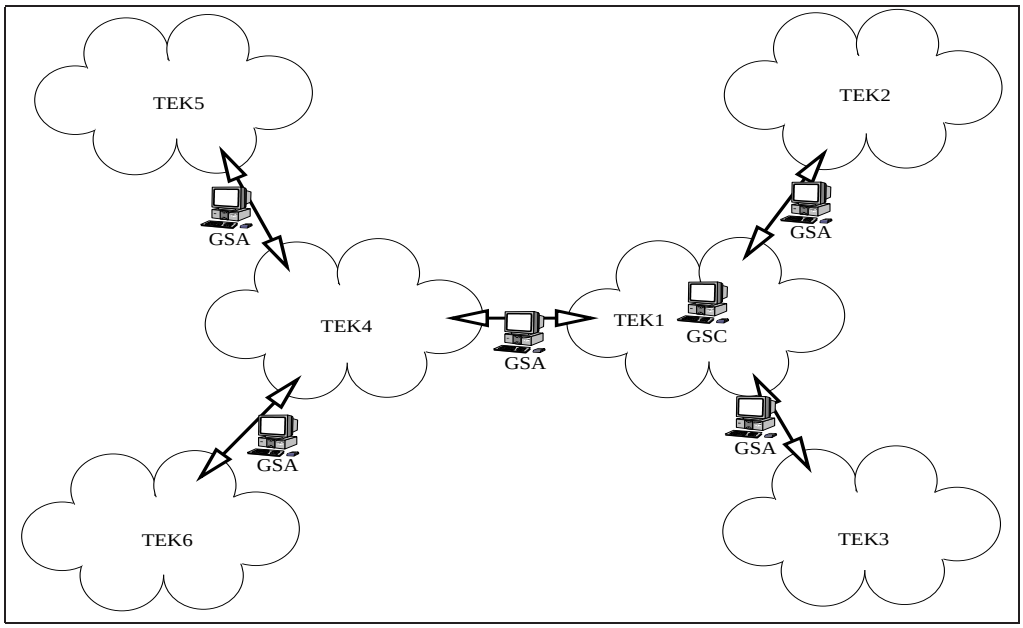


Figure 3.11: An example of a Iolus architecture

When a membership change occurs in a sub-group, only that sub-group is involved in a re-key process. This way, Iolus scales to large groups and mitigates *1-affects-n* phenomenon. However, Iolus has the drawback of affecting the data path. Indeed, there is a need for translating the data that goes from one sub-group, and thereby one key, to another. This induces decryption / re-encryption operations that are not tolerated by most of delay sensitive applications.

KEYED HIERARCHICAL MULTICAST PROTOCOL (KHIP): Shields et al. [127] proposed the Keyed HIerarchical multicast Protocol (KHIP). KHIP is based on a multicast tree built using OCBT [126] routing protocol, and uses a different TEK per each branch of the tree. It uses also an authentication service [59] based on certification to authenticate members and on-tree routers. The multicast tree is organized into sub-branches. Each sub-branch is managed by a trusted router which manages the TEK used within this sub-branch. When a source is ready to send a message to the group, it generates a random key K_T, encrypts the message with K_T, and encrypts K_T with the TEK of the sub-branch to which the source is attached. Then the source puts the encrypted K_T in the header of the packet carrying the message and multicasts the packet. The members located in the same sub-branch know the TEK of the sub-branch and hence can decrypt the K_T and then decrypt the message with K_T. When a border router of the sub-branch (a trusted router at the intersect between two sub-branches) receives the packet, it decrypts the K_T and re-encrypts it using the TEK of the adjacent sub-branch to which the so translated packet will be forwarded. This process is followed until the message reaches all the group members. When a new member joins a

sub-group, the router responsible for that sub-branch generates and distributes a new TEK for the sub-branch encrypted with the old one. When a member leaves a sub-branch, the corresponding router distributes a new TEK encrypted with the public key of each remaining member and signed with the router's private key.

Even thought KHIP reduces the decryption / re-encryption operations to a single key per packet, it still suffers from the delay variations of packet delivery due to these operations, and most of applications that require real-time transmission do not tolerate such delays.

CIPHER SEQUENCES (CS): Molva and Pannetrat [89] proposed a framework for multicast security that is based on Reversible Cipher Sequences. A function $f(S, a)$ is called Cipher Group (CG) if it has the following characteristics: there is a sequence of n elements $a_i (1 \leq i \leq n)$, and there is a sequence of $n+1$ elements S_i $(0 \leq i \leq n)$, such as $S_i = f(S_{i-1}, a_i)$ for $i > 0$ and S_0 is the initial value; and for every pair (i, j), where $i > j$, there exists a function $h_{i,j}$ such as $S_i = h_{i,j}(S_j)$. The multicast tree is rooted at the source, the group members are the leaves and internal nodes are intermediate elements of the multicast communication. Now, let S_0 be the message to be multicast and let every node N_i be assigned a value $a_i > 1$. When node N_i receives a value S_j from its parent N_j, it computes $S_i = f(S_j, a_i)$ and sends S_i to its children that can be either leaves or other internal nodes. The leaves are assigned the function $h_{0,n}$, which enables them to compute S_0 from S_n, since $S_0 = h_{0,n}(S_n)$, and therefore recover the original data. Figure 3.12 shows an example of Molva's scheme that may be described as:

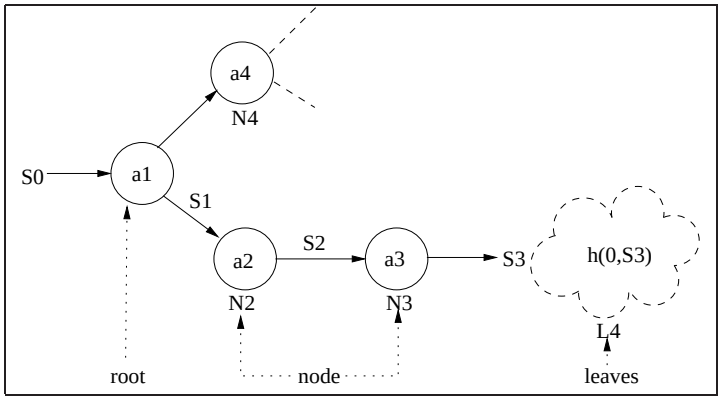


Figure 3.12: An example of Molva's scheme

1. the root calculates $S_1 = f(S_0, a_1)$ and sends S_1 to N_2 and N_4;
2. node N_2 calculates $S_2 = f(S_1, a_2)$ and sends S_2 to N_3;
3. node N_3 calculates $S_3 = f(S_2, a_3)$ and sends S_3 to leaves L_4;
4. leaves L_4 calculate $S_0 = h_{0,4}(S_3)$ and recover the original data: S_0.

When a membership change occurs in a leaf, the corresponding node N_n receives a new value a'_n and all members in the leaf receives a new function $h'_{0,n}$. Naturally, if the membership change

occurred because of member removal, the removed member will not receive the new $h'_{0,n}$, thus will not be able to recover S_0.

3.2.2 Time-driven re-keying

YANG ET AL. PROTOCOL: Yand et al. [143] proposed a reliable re-keying approach. In the proposed architecture, the multicast group is organized into a set of subgroups, where each subgroup is managed by a Key Server (KS). The role of a KS is to re-key the members of its subgroup periodically. In other words, the membership changes that occur during a specific period of time are batched, then the KS makes a single re-key that takes into consideration those membership changes. The overall KSs share a common *KS secret key*. When a KS receives a multicast message encrypted with its local TEK (sent by one of its subgroup members), it decrypts it and re-encrypts it using the *KS secret key*. Then, it multicasts the so re-encrypted message to the other KSs. In turn, these KSs decrypt the message using the *KS secret key* and re-encrypt it using their respective local TEKs. Then, each KS multicasts the so re-encrypted message to its subgroup.

SCALABLE INFRASTRUCTURE FOR MULTICAST KEY MANAGEMENT (SIM-KM): Mukherjee and Atwood [92] proposed a multicast key management infrastructure called SIM-KM: Scalable Infrastructure for Multicast Key Management. SIM-KM bases on subgrouping with message transformation by local controllers. In contrast to solutions based on subgrouping, SIM-KM uses *proxy encryption* [91] to transform data at the border of a subgroup. Proxy functions convert cipher text for one key into cipher text for another key without revealing secret decryption keys or clear text messages. This allows SIM-KM to do subgrouping with data transformation in order to limit the impact of re-keying, even thought intermediaries are not trusted entities.

3.2.3 Comparison

In table 3.4, we compare some of the above protocols against the following criteria:

1. *Key independence*: a protocol is said *key independent* if a disclosure of a key does not compromise other keys.
2. *1-affects-n*: a protocol suffers from the *1-affects-n* phenomenon if a single membership change in the group affects all the other group members.
3. *Local re-key*: membership changes in a sub-group should be treated locally.
4. *Data transformation*: data is transformed using some means when messages pass from a sub-group to another.

We notice that Iolus, Cipher Sequences, Yang et al. and SIM-KM protocols affect the data path when the messages pass through a subgroup. KHIP does not transform data itself but transforms the keys with which data is encrypted and hence reduces the delays induced by the transformations

Protocol	Key independence	1-affects-n	Local re-key	Data transformation
Iolus	Yes	No	Yes	Yes
KHIP	Yes	No	Yes	No
Cipher Sequences	Yes	No	No	Yes
Yang et al.	Yes	No	Yes	Yes
SIM-KM	No	No	Yes	Yes

Table 3.4: Comparison of some decentralized group key management protocols

at the borders of sub-groups. SIM-KM uses proxy encryption that allows to transform data without having to reveal encryption keys or clear text, and thereby the protocol is more suitable for situations where local controllers may be not trustworthy. The overall protocols succeed to mitigate the *1-affects-n* phenomenon, since they use an independent TEK per sub-group, and hence a membership change in a subgroup affects only members belonging to the same subgroup. However, reducing *1-affects-n* phenomenon is not for free: the multicast messages (or keys for KHIP) should be transformed by the Local sub-group controllers whenever they pass through a new subgroup. These transformations affect the packet delivery delays, and unfortunately this is not suitable for many applications that require real-time transmission.

3.2.4 Conclusion

In this chapter we have presented a state of the art of *group key management*. We have classified existing solutions into two main categories: the *common TEK approach* and the *TEK per subgroup approach*. Throughout this chapter, we refined this classification according to the common concept and techniques used by the proposed solutions. We illustrated each identified sub-category with relevant solutions from the literature, and we compared them against pertinent performance criteria. We showed that both proposed approaches suffer from great concerns depending on group dynamism: the *common TEK approach* suffers from the *1-affects-n* phenomenon, where a single group membership change (join or leave) results in a re-keying process that disturbs all group members to update the TEK. Moreover, centralized protocols are not scalable, and distributed ones bring new challenges such as synchronization and conflict resolution. *Time-driven re-keying protocols* attempt to reduce the *1-affects-n* phenomenon by batch re-keying, but then cannot be used with critical applications that require to take into consideration the membership change instantly. On the other hand, the *TEK per subgroup approach* reduces the *1-affects-n* problem. This is advantageous for highly dynamic multicast groups. However, this approach requires transformation of sent messages whenever they pass from a sub-group to another, and this may not be tolerated by applications that are sensitive to packet delivery delay variations. Besides, this approach would not be worthy with relatively static groups because the multiple transformations would induce avoidable delays and useless computation overheads. These shortcomings are due to the lack of *dynamism awareness* in existing group key management schemes.

In the following chapter, we propose the Scalable and Adaptive Group Key Management approach (SAKM) which is a *dynamism aware* scheme that takes advantage of the both discussed approaches

by dynamically adapting the key management process with respect to the frequency of membership changes.

Chapter 4

Scalable and Adaptive Group Key Management

WE showed, in the previous chapter, that current group key management proposals do not scale well with large and highly dynamic groups, either because of the *1 affects n* problem or because of the *excess* of *data transformation* operations problem. In this chapter we propose a new *Scalable and Adaptive Key Management scheme (SAKM)* that addresses these two problems by taking into consideration the dynamic aspect of the group members. We tackle the scalability issue by organizing the multicast group into clusters, where each cluster uses its own TEK. In contrast to existing solutions that use this concept, with SAKM: the organization of the group into clusters is updated periodically depending on the *dynamism* of the members during the secure session. The partitioning aims to minimize both data transformation (decryption / re-encryption in our case) and re-keying overheads depending on the membership change behavior. Indeed, many studies [2] show that the membership behavior of group members in multicast sessions is likely to be not uniform through a large scale group and during the whole session. Some parts of the network may be more dynamic than others during some periods of time and become more stable afterward. In this case, it would be interesting to use a protocol that restricts the re-key to the areas with frequent membership changes. Thereby, SAKM is very efficient in such situations. Simulation results show that SAKM scales well to large groups by minimizing the *1-affects-n* phenomenon, while it reduces the decryption / re-encryption operations thanks to the *adaptive* dimensioning of the clusters depending on the membership dynamism. Note that we proposed also another protocol called *Adaptive Key Management Protocol (AKMP)* in [12]. *AKMP* bases also on an adaptive scheme, but operates at the routing level. This is a promising solution for secure group communication in *ad hoc* networks where all the group members are highly dynamic and involved in the routing function. Indeed, a research team at INRIA research institute has undertaken *AKMP* and adapted it for secure group communication in the context of *ad hoc networks* [17]. This chapter relies on [24, 28], and focuses only on *SAKM* which is more elaborated and better modeled.

The remaining of this chapter is organized as follows : In the following section, we give an overview of the proposed architecture, then we present the analytic model on which relies the SAKM protocol in section 2. In section 3 we give a formalization of the SAKM solution. Finally we present details of the SAKM protocol in section 4 and the simulation results in section 5.

4.1 Overview of SAKM Architecture

We tackle the *1-affects-n* phenomenon and thereby the *scalability issue* by organizing the multicast group into a *hierarchy of subgroups*. Each subgroup is managed by a local controller, that we call *SAKM-Agent*. Each *SAKM-agent* is a member in its subgroup and in its parent subgroup. Thus, a *SAKM-agent* plays the role of a *proxy* for its subgroup in the parent subgroup. When a *SAKM-agent* receives a multicast message from the parent subgroup, it forwards it to the subgroup under its control. Initially, all the subgroups use the same *TEK*, and thereby the task of the *SAKM-agents* would be simply to forward received multicast messages to their subgroups. In this case, we say that the *SAKM-agent*s are in a *passive* state. In this configuration, a single membership change in any subgroup would induce the distribution of a new TEK to the overall subgroups, because of using a *common* TEK. Hence, when one of the subgroups becomes *dynamic*, we would like to be able to isolate that subgroup from the other subgroups in order to restrict the impact of re-keying to that dynamic subgroup, and thereby reducing the *1-affects-n* phenomenon. We propose in our scheme, that in such a situation, the *SAKM-Agent* of the dynamic subgroup takes the decision to use an *independent* TEK for its subgroup. Therefore, upon receiving multicast messages from the parent subgroup (encrypted using the parent's TEK), the *SAKM-Agent* will translate them to the TEK used within its subgroup before forwarding them downward. In this case, we say that the *SAKM-Agent* is in an *active* state. Our objective is to design an *adaptive* architecture where the different *SAKM-Agent*s *switch* from a state to another depending on the faced dynamism in their subgroups and taking into consideration the induced re-keying and translation overheads. The problem of affecting a state (*active* or *passive*) to the *SAKM-Agent*s is equivalent to partitioning the whole hierarchy of subgroups into clusters of sub-hierarchies using *common* TEKs, where the root of each cluster is a *active SAKM-Agent* and internal *SAKM-Agent*s of a cluster are *passive*. Figure 4.1 illustrates the different components of our architecture. Without loss of generality, we use decryption / re-encryption as a means of data translation in our case. Our proposal can be easily adapted to use other data translation techniques proposed in the literature, such as: cipher sequences [89], random key decryption / re-encryption [127], proxy encryption [91] etc.

Periodically, SAKM Agents exchange dynamism information about their subgroups. Based on these information, each agent estimates two costs: the first cost is the overhead cost induced if the agent becomes *active* and the second cost is the cost induced if the agent becomes *passive*. By comparing the two costs, the SAKM Agent decides whether to become *active* or *passive* (i.e. if the first cost is lower then the agent becomes active, else it becomes passive). If an agent becomes *passive* it *merges* with its parent cluster and so it uses its parent TEK. If an agent becomes *active* it forms a new *separate* cluster and so uses an independent local TEK. After each periodic information exchange about subgroups' dynamism, we may obtain a new partition of the group into clusters due to *split* and/or *merge* operations. This new partition suites better the current membership behavior of the group in terms of both decryption / re-encryption cost and *1 affects n* overhead. Hence, SAKM approach offers an efficient and adaptive scheme that maintains good performance during the whole secure multicast session. In the following sections, we give detailed

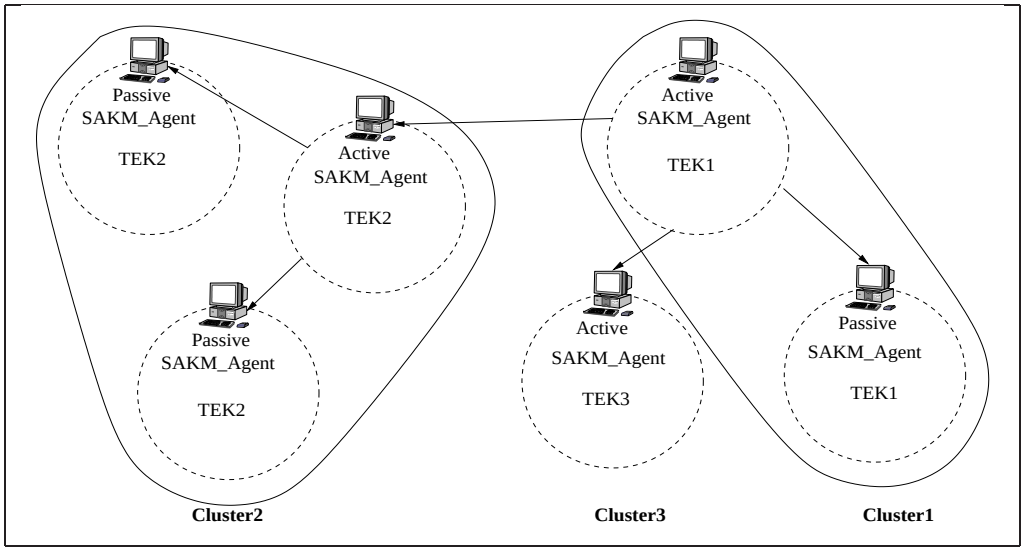


Figure 4.1: SAKM Architecture

description of the SAKM approach by presenting an analytic model and a set of theoretical results on which rely the heuristic and protocols presented in section 4.4.

4.2 SAKM Analytic Model

In this section we present an analytic model of an elementary system composed of two SAKM subgroups i and j. j is the parent subgroup of i. Each subgroup is managed by a SAKM agent. By presenting the model, we aim to find out the criterion which should be used by an agent to decide whether it has to be in *active* or *passive* state. The two situations: *active* agent (or *split* subgroups) and *passive* agent (or *merged* subgroups) induce different overheads regarding decryption/re-encryption operations and re-keying messages. By comparing the overheads in the two situations, the agent makes the best decision.We use these results as building blocks to propose the Scalable and Adaptive Key Management Protocol for secure group communication in section 4.4

4.2.1 Preliminaries and nomenclature

In what follows, we will quantify the overhead induced by decryption/re-encryption and re-keying in two cases:

- **case 1:** in the case where the two subgroups i and j are merged to use the same keying material. We denote the induced overhead in this case by $(O_{i,j}^{(m)})$;
- **case 2:** in the case where the two subgroups i and j are split into two different clusters, and hence each of them uses its own keying material. We denote the induced overhead in this

case by $(O_{i,j}^{(s)})$.

Thus, a SAKM Agent evaluates and compares the two quantities $O_{i,j}^{(m)}$ and $O_{i,j}^{(s)}$. If $O_{i,j}^{(m)} > O_{i,j}^{(s)}$ then SAKM agent becomes *active* (i.e. it is more efficient to separate the subgroup from its parent subgroup so that each of them uses its own keying material: a split operation), else it becomes *passive* (i.e. it is more efficient to merge the subgroup with its parent subgroup so that they use the same keying material: a merge operation). In this section, we calculate the two overheads: $O_{i,j}^{(m)}$ and $O_{i,j}^{(s)}$.

Let $C_{i,j}^{(m)}$ be the average cost of re-keying in the case i and j are merged (case 1). And $C_{i,j}^{(s)}$ the average cost of re-keying in the case i and j are split (case 2). Suppose that the decryption / re-encryption overhead (per time unit) depends only on the encryption system (τ), the computation power of the agent that does the operation (P_i) and the rate of messages (r) that characterizes the application. We denote the whole decryption/re-encryption overhead by $\tau(P_i, r)$. Table 4.1 gives the nomenclature used to present the analytic model.

Symbol	Signification
$O_{i,j}^{(m)}$	The overhead induced by merging subgroups i and j
$O_{i,j}^{(s)}$	The overhead induced by splitting subgroups i and j
r	Message arrival rate
P_i	Computation power of SAKM agent of subgroup i
$\tau(P_i, r)$	Decryption / re-encryption overhead (per time unit) induced by decrypting and re-encrypting messages that arrive at a rate r using a crypto-system τ processed by an agent whose power is P_i
$C_{i,j}^{(m)}$	Cost of re-keying in case subgroups i and j are merged
$C_{i,j}^{(s)}$	Cost of re-keying in case subgroups i and j are split
$E[C_{\lambda,\mu}]$	Expected number of re-key messages per unit of time in a system where members arrive at a rate λ and stay in the group $1/\mu$ units of time in the average
$E[J_i]$	Expected number of re-key messages per unit of time due to a join in a system that contains i concurrent members
$E[L_i]$	Expected number of re-key messages per unit of time due to a leave in a system that contains i concurrent members
$M_{\{i,j\}}$	Mutual impact re-keying cost due to merging subgroups i and j

Table 4.1: Nomenclature

The overhead induced by *split* subgroups is the sum of both re-keying and decryption/re-encryption overheads, thus

$$O_{i,j}^{(s)} = C_{i,j}^{(s)} + \alpha\tau(P_i, r) \tag{4.1}$$

Where (α) is the factor characterizing the weight given to a decryption/re-encryption operation compared to a re-key message. This parameter will be further discussed in the following paragraphs.

The overhead induced by merging subgroups corresponds to the re-keying overhead alone since there is no decryption/re-encryption in the case of merged subgroups, and thus

$$O_{i,j}^{(m)} = C_{i,j}^{(m)} \tag{4.2}$$

In order to approximate $C_{i,j}^{(m)}$ and $C_{i,j}^{(s)}$, we present first the following multicast dynamism model.

4.2.2 Multicast Dynamism Model

Almeroth et al. showed in [2, 3] that the dynamism of some multicast sessions over the MBone can be modeled as follows: the users arrive in a multicast group according to a Poisson process with rate λ (arrivals/time unit), and the membership duration of a member in the group follows an exponential distribution with a mean duration $\frac{1}{\mu}$ time units. The average number of concurrent users in a subgroup is given by $\frac{\lambda}{\mu}$. In our case, we apply this model to each subgroup. Unlike [32], we do not suppose that the subgroups are likely to be joined by the members. Instead, each subgroup is characterized by its own parameters λ and μ. Moreover, we suppose that the parameters λ and μ change over time and thus each SAKM agent adjusts its estimations of λ and μ every θ time units in order to approximate better the re-keying overhead. Each subgroup can, therefore, be modeled by a Markov process [32]. Let $Q = \{0, 1, 2, 3, \ldots\}$ denote the system state corresponding to the number of concurrent users in a subgroup. Let π_k be the steady state probability that $Q = k$. It is well known [75] that

$$\pi_k = \frac{(\frac{\lambda}{\mu})^k}{k!} \cdot e^{-(\frac{\lambda}{\mu})} \tag{4.3}$$

Let J_k and L_k be the induced costs when a user joins and leaves the subgroup in state k, respectively. Note that J_k and L_k are random variables depending on where the user joins or leaves the subgroup. Let $E[J_k]$ and $E[L_k]$ be the expected values of J_k and L_k, respectively. We denote by $E[C_{\lambda,\mu}]$ the expected number of re-key messages per time unit. By the steady state properties of the Markov chain, $E[C_{\lambda,\mu}]$ can then be expressed by

$$E[C_{\lambda,\mu}] = \lambda \sum_{k=0}^{\infty} \pi_k (E[J_k] + E[L_k]) \tag{4.4}$$

In order to calculate $E[C_{\lambda,\mu}]$, we need to simplify the expression of π_k by approximation. Authors in [32] showed that approximating π_k as a δ-function at its mean is a good approximation, i.e.,

$$\pi_k \approx \delta\left(k - \frac{\lambda}{\mu}\right) \tag{4.5}$$

where

$$\delta\left(k - \frac{\lambda}{\mu}\right) = \begin{cases} 1 \text{ if } k = \frac{\lambda}{\mu} \\ 0 \text{ otherwise} \end{cases} \tag{4.6}$$

Therefore

$$\begin{aligned} E\left[C_{\lambda,\mu}\right] &\approx \lambda \sum_{k=0}^{\infty} \delta\left(k - \frac{\lambda}{\mu}\right)\left(E[J_k] + E[L_k]\right) \\ &= \lambda(E[J_{\frac{\lambda}{\mu}}] + E[L_{\frac{\lambda}{\mu}}]) \end{aligned} \tag{4.7}$$

In case of split subgroups, each membership change implies a re-key process in the subgroup where the membership change occurs and thus we can approximate $C_{i,j}^{(s)}$ by

$$C_{i,j}^{(s)} \approx E[C_{\lambda_i,\mu_i}] + E[C_{\lambda_j,\mu_j}] \tag{4.8}$$

and hence, from (4.1) and (4.8), we obtain

$$O_{i,j}^{(s)} \approx E[C_{\lambda_i,\mu_i}] + E[C_{\lambda_j,\mu_j}] + \alpha\tau(P_i, r) \tag{4.9}$$

In case of merged subgroups, each membership change implies a re-key process in both subgroups that share the same TEK. We say that a membership change that occurs in subgroup i has an impact on subgroup j (which is merged with i), and a membership change that occurs in subgroup j has an impact on subgroup i. Therefore, there is a *mutual impact re-keying* in case of merged subgroups because of sharing a same TEK. Now, we define some terminology to simplify the following discussion:

Definition 1 *Consider a SAKM-cluster C. We mean by mutual impact re-keying in C, the sum of required re-key messages sent by SAKM-agents of the SAKM-subgroups $C - \{i\}$ when a membership change occurs in subgroup i (for each $i \in C$). We denote by $E\left[M_C\right]$, the expected number of mutual impact re-keying messages per time unit. In what follows, we call $E\left[M_C\right]$ the mutual impact re-keying cost of cluster C.*

In our case, the cluster contains two merged subgroups i and j. Thus, according to this definition, we approximate $C_{i,j}^{(m)}$ by

$$C_{i,j}^{(m)} \approx E[C_{\lambda_i,\mu_i}] + E[C_{\lambda_j,\mu_j}] + E[M_{\{i,j\}}] \tag{4.10}$$

and hence, from (4.2) and (4.10), we obtain

$$O_{i,j}^{(m)} \approx E[C_{\lambda_i,\mu_i}] + E[C_{\lambda_j,\mu_j}] + E[M_{\{i,j\}}] \tag{4.11}$$

According to (4.9) and (4.11), we notice that to compare $O_{i,j}^{(m)}$ and $O_{i,j}^{(s)}$, it is sufficient to compare the two quantities:

$$E[M_{\{i,j\}}] \tag{4.12}$$

and

$$\alpha\tau(P_i, r) \tag{4.13}$$

In conclusion, we notice that $O_{i,j}^{(s)}$ (equation 4.9) is equal to the expected number of re-key messages per time unit in each subgroup plus the overhead induced by decryption / re-encryption operations (cf. figure 4.2). $O_{i,j}^{(m)}$ (equation 4.11) is equal to the number of re-key messages per

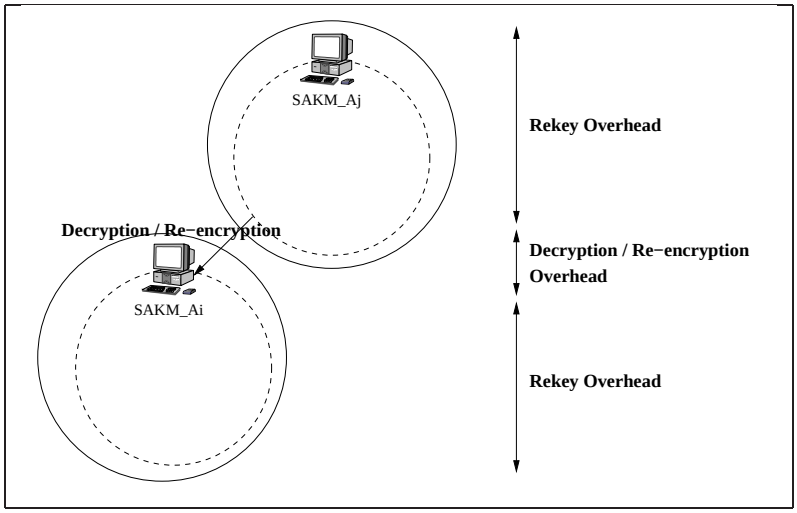


Figure 4.2: Overhead induced by split subgroups

time unit in each subgroup plus the impact (in term of messages per time unit) of each subgroup on the other subgroup in case of a merge (cf. figure 4.3). As the number of re-key messages in each subgroup is common to the two situations, it suffices to compare the differences, i.e. the quantities given by the formulas (4.12) and (4.13).

Now we can see that the parameter α plays a key role in the operation and the performance of the proposed scheme. Indeed, the parameter α allows to trade *the application level requirement in term of synchronization between the source and receivers*, for *minimizing the 1-affects-n phenomenon*. In other words:

- If the application requires *high synchronization between the source and receivers* (such as videoconferencing), then we give to α a *large* value, so that decryption / re-encryption operations will be minimized by favoring merging over splitting.
- If the application induces *high dynamism* (such as video on demand), we give to α a *small* value , so that *1-affects-n* phenomenon will be minimized by favoring splitting over merging.

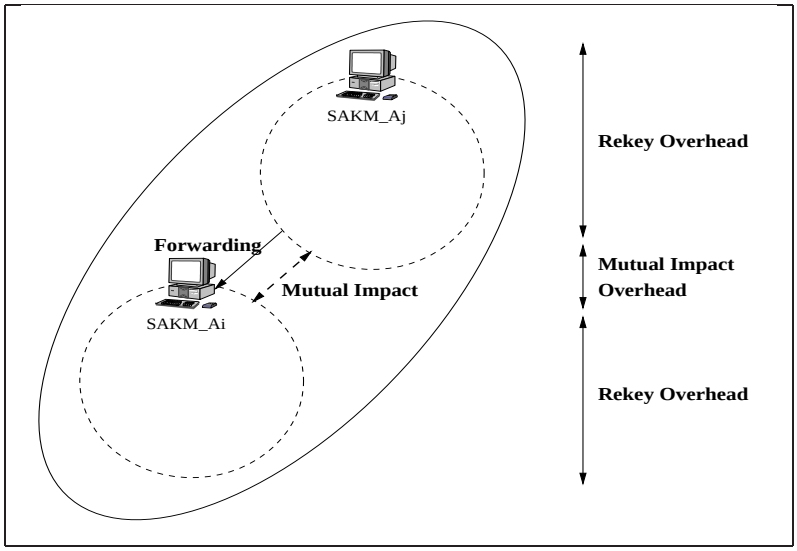


Figure 4.3: Overhead induced by merged subgroups

4.2.3 Application of the analytic model to actual re-key strategies

The re-key overhead depends on the used re-key strategy. As SAKM is an open architecture, each SAKM agent is free to use the most suitable re-key strategy for its subgroup. For clarity reasons, we suppose that all SAKM agents use the same re-key strategy. In order to give illustrative examples we consider mainly two strategies: the "n root/leaf pairwise" [135] protocol and the "Logical Key Hierarchy (LKH)" [136] protocol. For each of these two strategies we compute the re-keying and the mutual impact re-keying costs in both situations : split and merge of two subgroups. We use these costs to compute $O_{i,j}^{(m)}$ and $O_{i,j}^{(s)}$ for each strategy.

re-keying cost

The n root/leaf pairwise approach: In this approach, all the members share the same TEK. When a member joins the group, the agent multicasts the new TEK (encrypted with the old one) to the old members and unicasts it to the new member encrypted with the secret key that it shares with it. Thus, the cost of a join re-key is two messages

$$E\left[J_{\frac{\lambda}{\mu}}\right] = 2 \tag{4.14}$$

In case of a leave, the old TEK is compromised and hence the agent sends the new TEK to each remaining member encrypted with the key that it shares with it, and hence the number of messages in a leave re-key is equal to the size of the group. Thus,

$$E\left[L_{\frac{\lambda}{\mu}}\right] = \frac{\lambda}{\mu} \tag{4.15}$$

The logical key hierarchy approach: If we approximate the key graph as a full tree at any time (with degree d): in case of a join, the agent should redistribute each key that is on the path from the leaf that represents the secret key associated to the new member to the root which represents the TEK. Thus, each key from the leaf of the new member to the root (there are $\log_d \frac{\lambda}{\mu}$ nodes) is sent twice: it is sent to the new member by unicast encrypted with the child key that is known by the new member, and sent by multicast to the members that share the node encrypted with its old version. Therefore, we obtain

$$E\left[J_{\frac{\lambda}{\mu}}\right] = 2\log_d \frac{\lambda}{\mu} \tag{4.16}$$

In case of a leave, the keys on the path from the leaf that represents the leaving member to the root are compromised and hence each of them should be updated. Each key from the leaf (of the leaving member) to the root (there are $\log_d \frac{\lambda}{\mu}$ nodes) is sent by multicast (encrypted with its d child keys) to the residual members that share its d child keys. Thus

$$E\left[L_{\frac{\lambda}{\mu}}\right] = d\log_d \frac{\lambda}{\mu} \tag{4.17}$$

Mutual impact re-keying cost

To illustrate the computation of the *mutual impact re-keying cost*, we consider the two subgroups i and j and we compute the mutual impact re-keying cost in the case when these subgroups are merged into the same cluster. In what follows, l designates either i or j, and $\bar{l}$ designates the counterpart of l (i.e. if $l = i$ then $\bar{l} = j$ and vice versa).The member arrival rate at subgroup l is λ_l. Let IJ_k^l and IL_k^l be the impact costs at subgroup l, in state k when a member joins or leaves subgroup $\bar{l}$, respectively. By the steady state properties of the Markov Chain,

$$E\left[M_{\{i,j\}}\right] = \lambda_j \sum_{k=0}^{\infty} \pi_k (E\left[IJ_k^i\right] + E\left[IL_k^i\right]) + \lambda_i \sum_{k=0}^{\infty} \pi_k (E\left[IJ_k^j\right] + E\left[IL_k^j\right]) \tag{4.18}$$

When a member joins subgroup $\bar{l}$, SAKM-agent of subgroup l sends the received new TEK by multicast (one message) to its members encrypted with the old TEK which is not compromised. Hence, $IJ_k^l = 1$. Thus, equation 4.18 becomes

$$E\left[M_{\{i,j\}}\right] = \lambda_j \sum_{k=0}^{\infty} \pi_k (1 + E\left[IL_k^i\right]) + \lambda_i \sum_{k=0}^{\infty} \pi_k (1 + E\left[IL_k^j\right]) \tag{4.19}$$

By considering the approximation of $\pi_k \approx \delta\left(k - \frac{\lambda_l}{\mu_l}\right)$ [32], equation 4.19 becomes

$$E\left[M_{\{i,j\}}\right] = \lambda_j \left(1 + E\left[IL^i_{\frac{\lambda_i}{\mu_i}}\right]\right) + \lambda_i \left(1 + E\left[IL^j_{\frac{\lambda_j}{\mu_j}}\right]\right) \tag{4.20}$$

The computation of $E\left[IL^l_{\frac{\lambda_l}{\mu_l}}\right]$ depends on the actual re-key strategy:

The n root/leaf pairwise approach: In case of a leave from subgroup $\bar{l}$, the old TEK shared between subgroups i and j, is compromised and hence the SAKM-agent of subgroup l sends the new TEK encrypted with each secret key that it shares with its $\frac{\lambda_l}{\mu_l}$ members. Therefore, with this re-key strategy

$$E\left[IL^{l}_{\frac{\lambda_l}{\mu_l}}\right] = \frac{\lambda_l}{\mu_l} \tag{4.21}$$

The logical key hierarchy approach: We approximate the key tree as a full tree at any time (with degree d). In case of a leave at subgroup $\bar{l}$, the old TEK is compromised and hence the agent sends the new TEK by multicast and encrypted with each root's child key (there are d child keys: the degree of the tree) that are shared by the members of subgroup l. Therefore, with this re-key strategy

$$E\left[IL^{l}_{\frac{\lambda_l}{\mu_l}}\right] = d \tag{4.22}$$

Table 4.2 summarizes the overheads and the quantities to be compared in both strategies.

	N root/leaf pairwise	*Hierarchical key graph*
$O^{(m)}_{i,j}$: *the whole overhead in merged subgroups*	$(\lambda_i + \lambda_j)\left(3 + \frac{\lambda_i}{\mu_i} + \frac{\lambda_j}{\mu_j}\right)$	$\lambda_i\left(1 + d + (d+2)\log_d \frac{\lambda_i}{\mu_i}\right)$ $+\lambda_j\left(1 + d + (d+2)\log_d \frac{\lambda_j}{\mu_j}\right)$
$O^{(s)}_{i,j}$: *the whole overhead in split subgroups*	$\lambda_i\left(2 + \frac{\lambda_i}{\mu_i}\right) +$ $\lambda_j\left(2 + \frac{\lambda_j}{\mu_j}\right) + \alpha\tau(P_i + r)$	$\lambda_i(d+2)\log_d \frac{\lambda_i}{\mu_i}$ $+\lambda_j(d+2)\log_d \frac{\lambda_j}{\mu_j}$ $+\alpha\tau(P_i + r)$
Mutual impact overhead	$\lambda_i\left(1 + \frac{\lambda_j}{\mu_j}\right) + \lambda_j\left(1 + \frac{\lambda_i}{\mu_i}\right)$	$\lambda_i\,(1+d) + \lambda_j\,(1+d)$
Decryption re-encryption overhead	$\alpha\tau(P_i + r)$	$\alpha\tau(P_i + r)$

Table 4.2: SAKM Adaptability regarding re-key strategy

4.3 SAKM problem statement

In the above sections, we discussed the overhead induced by merging two subgroups. In a general case, we deal with merging many SAKM subgroups whenever splitting them is more expensive than merging them together to use the same keying material. According to definition 1 of mutual impact re-keying, we have $M_C = \sum_{i \neq j}^{i,j \in C} M_{\{i,j\}}$, where C is a cluster of merged SAKM-subgroups, and the expected number of mutual impact re-keying messages per time unit can be expressed by

$$E\left[M_C\right] = \sum_{i \neq j}^{i,j \in C} E\left[M_{\{i,j\}}\right] \tag{4.23}$$

4.3.1 Illustrative example

Let $G = \{x, y, z\}$, where x, y and z are SAKM subgroups and form a virtual hierarchy (cf. figure 4.4.a). To this tree, we can associate four partitions $\{x, y, z\}$, $\{\{x, y\}, z\}$, $\{\{x, z\}, y\}$ and

$\{\{x, y, z\}\}$ (cf. figure 4.4). The cost associated to the partition $\{\{x, y\}, z\}$ is the one induced by the decryption / re-encryption overhead at z and the mutual impact re-keying overhead due to merging x and y and thus the cost associated to the partition $\{\{x, y\}, z\}$ is

$$E\left[M_{\{x,y\}}\right] + \alpha\tau(P_z, r)$$

In the same way, we establish the table 4.3 which associates a cost to each G-partition.

Partition	*Cost*
$\{x, y, z\}$	$\alpha\tau(P_y, r) + \alpha\tau(P_z, r)$
$\{\{x, y\}, z\}$	$E\left[M_{\{x,y\}}\right] + \alpha\tau(P_z, r)$
$\{\{x, z\}, y\}$	$E\left[M_{\{x,z\}}\right] + \alpha\tau(P_y, r)$
$\{\{x, y, z\}\}$	$E\left[M_{\{x,y,z\}}\right]$

Table 4.3: Costs associated to G-partitions

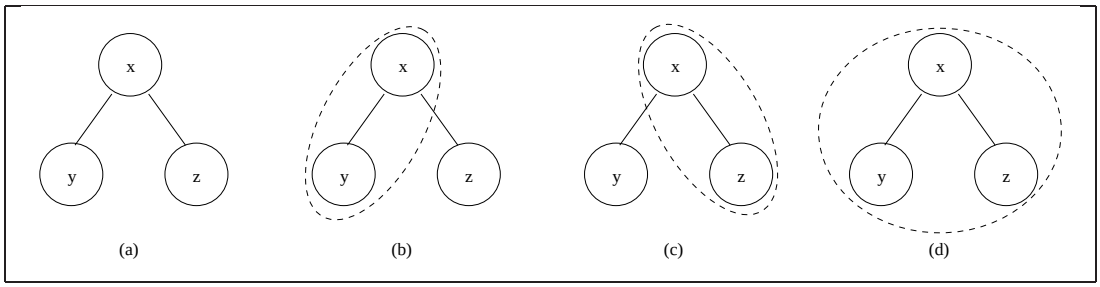


Figure 4.4: Possible G-partitions

SAKM aims to partition G into the configuration that induces the minimum cost.

4.3.2 SAKM problem formalization

The SAKM problem can be formalized as a Tree Partitioning Problem (TPP)[48] as follows: we associate to a SAKM hierarchy a tree structure $T = (G, V)$, where G is the set of SAKM-subgroups and V the set of edges that identify *adjacency* relations between subgroups. $T_i = (G_i, V_i)$ is said to be a subtree of T if T_i is a tree such that $G_i \subset G$, $G_i \neq \Phi$ and $V_i \subset V$. Let $E = \{T_1, T_2, \ldots T_i, \ldots\}$ be the family of all possible subtrees of T. These subtrees constitute the SAKM clusters $\{G_1, G_2 \ldots, G_i, \ldots\}$ of the given SAKM hierarchy. To each cluster G_i, we associate a cost $C(i)$ which is the sum of decryption / re-encryption cost at the G_i's root and the mutual impact re-keying cost due to merging SAKM subgroups into the cluster G_i.

$$C(i) = E\left[M_{G_i}\right] + \alpha\tau(P_z, r), \text{ with } z \text{ being the } G_i\text{'s root.}$$

We are interested in finding a sub-family of E: $F = \{T_{i_1}, T_{i_2}, \ldots, T_{i_i}, \ldots\}$, so that the corresponding SAKM clusters $\{G_{i_1}, G_{i_2}, \ldots, G_{i_i}, \ldots\}$ form a partition of G with a total minimal cost.

In other words, we want to:

$$\begin{cases} \min \sum_{G_i/T_i \in F} C(i) \\ \text{with } F \subset E \\ \text{and } \bigcup_{G_i/T_i \in F} G_i = G \\ \text{and } \forall G_i, G_j/T_i, T_j \in F, G_i \cap G_j = \Phi \text{ if } i \neq j \end{cases}$$

The following lemma gives the *size* of the SAKM problem.

Lemma 1 *If $\|G\| = p$ then the number of G-partitions is 2^{p-1}.*

Proof Each G-partition cluster is made up of a root SAKM subgroup whose SAKM agent is *active* and child subtrees of SAKM subgroups whose SAKM agents are *passive*. Thus, each G-partition defines a combination of *active* / *passive* SAKM agents. And, vice versa, each combination of *active* / *passive* SAKM agents (except the root which should be always *active*) defines a G-partition. It follows that the number of possible combinations (of $p-1$ SAKM agents which can hold either *passive* or *active* state) is 2^{p-1}, which corresponds to the number of possible G-partitions.

4.4 SAKM Protocol

In order to find out the optimal G-partition, a naive solution would be to enumerate the 2^{p-1} possible configurations and to pick up the one with the smallest cost. This would induce a $O(2^{p-1})$ overhead which is not acceptable. In our case, we propose a heuristic to approach the optimal configuration with reasonable delays and overheads. The heuristic used by SAKM to partition the virtual hierarchy into clusters with independent keying material relies on the following results.

Lemma 2 *Let x and y be SAKM subgroups, with x parent of y. If $\alpha\tau(P_y, r) \leq E[M_{\{x,y\}}]$, then $\alpha\tau(P_y, r) + E\left[M_{\{x\} \cup Z_x}\right] + E\left[M_{\{y\} \cup Z_y}\right] < E\left[M_{\{x,y\} \cup Z_x \cup Z_y}\right]$, with Z_x any set of SAKM subgroups that can be merged with x and Z_y any set of SAKM subgroups that can be merged with y (cf. figure 4.5).*

Proof This lemma states that if the decryption / re-encryption cost at a SAKM agent y is smaller than the mutual impact re-keying cost induced by merging y with its parent subgroup x, then the overhead induced by splitting x and y stays less important than the overhead induced by merging many subgroups that include x and y. In fact, we have $E\left[M_{\{x,y\} \cup Z_x \cup Z_y}\right] = E\left[M_{\{x\} \cup Z_x}\right] +$

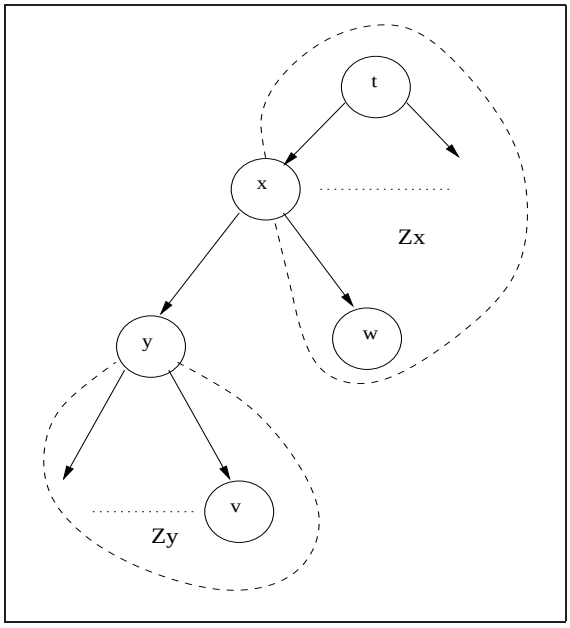


Figure 4.5: Lemma 2 illustration

$E\left[M_{\{y\}\cup Z_y}\right] + \sum_{\substack{i\in\{x\}\cup Z_x \\ j\in\{y\}\cup Z_y}} E\left[M_{\{i,j\}}\right]$ It follows then, that if $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right]$ Then

$$\begin{aligned}
&\alpha\tau(P_y, r) + E\left[M_{\{x\}\cup Z_x}\right] + E\left[M_{\{y\}\cup Z_y}\right] \\
&\leq E\left[M_{\{x,y\}}\right] + E\left[M_{\{x\}\cup Z_x}\right] + E\left[M_{\{y\}\cup Z_y}\right] \\
&< E\left[M_{\{x,y\}}\right] + E\left[M_{\{x\}\cup Z_x}\right] + E\left[M_{\{y\}\cup Z_y}\right] \\
&+E\left[M_{\{x\}\cup Z_y}\right] + E\left[M_{\{y\}\cup Z_x}\right] + \sum_{\substack{i\in Z_x \\ j\in Z_y}} E\left[M_{\{i,j\}}\right] \\
&= E\left[M_{\{x\}\cup Z_x}\right] + E\left[M_{\{y\}\cup Z_y}\right] + \sum_{\substack{i\in\{x\}\cup Z_x \\ j\in\{y\}\cup Z_y}} E\left[M_{\{i,j\}}\right] \\
&= E\left[M_{\{x,y\}\cup Z_x\cup Z_y}\right]
\end{aligned}$$

Corollary 1 *Let x and y be SAKM subgroups with x parent of y. If $\alpha\tau(P_y, r) \leq E\left[M_{x,y}\right]$, then y would be a cluster's root in the optimal SAKM-partition.*

Proof Suppose that $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right]$. If y is not a cluster's root in the optimal partition, then it should be an internal node to a cluster in the optimal partition. Let $\{x, y\} \cup Z_x \cup Z_y$ be that cluster (cf. figure 4.5). x is parent of y. Z_x and Z_y are SAKM subgroups' sets. The cost associated to this cluster in the optimal partition is: $E\left[M_{\{x,y\}\cup Z_x\cup Z_y}\right] + \alpha\tau(P_t, r)$ where t is the Z_x's root. Now, we show that if $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right]$ then, there exists a partition with a smaller overhead than the one that contains the cluster $\{x, y\} \cup Z_x \cup Z_y$.

In fact, if we split this cluster into two clusters by making y's SAKM agent *active*, we would have two clusters with a total cost of $\alpha\tau(P_y, r) + E\left[M_{\{y\}\cup Z_y}\right] + \alpha\tau(P_t, r) + E\left[M_{\{x\}\cup Z_x}\right]$.

According to lemma 2, we have

$$\alpha\tau(P_y, r) + E\left[M_{\{y\}\cup Z_y}\right] + E\left[M_{\{x\}\cup Z_x}\right] < E\left[M_{\{x,y\}\cup Z_x\cup Z_y}\right]$$

It follows that

$$\alpha\tau(P_y, r) + E\left[M_{\{y\}\cup Z_y}\right] + E\left[M_{\{x\}\cup Z_x}\right] + \alpha\tau(P_t, r) < E\left[M_{\{x,y\}\cup Z_x\cup Z_y}\right] + \alpha\tau(P_t, r)$$

which means that we found a partition with a cost smaller than the one pretended to be optimal.

Corollary 2 *Let x and y be SAKM subgroups with x parent of y. If $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right]$ then the y's SAKM agent can decide to become active and this would be an optimal decision.*

Proof If $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right]$ then it follows from corollary 1 that y must be a cluster's root in the optimal partition, which is equivalent to say that y must be *active* (assures decryption / re-encryption in the optimal configuration).

4.4.1 Overview of SAKM protocol

We remind that the SAKM architecture is made up of an hierarchy of multicast subgroups. The whole hierarchy forms a SAKM multicast group. The keying material inside a subgroup is managed by a SAKM agent. Two adjacent subgroups may merge to use the same parent's keying material if the mutual impact re-keying cost is less than the decryption / re-encryption cost at the child SAKM agent. In the contrary case, the two adjacent subgroups are split and each of them uses its own keying material. The SAKM aim is partitioning the hierarchy into clusters so that both decryption / re-encryption and re-keying overheads are minimized. Each cluster is a set of SAKM subgroups using the same keying material. SAKM uses a heuristic to approach the optimal solution. Corollary 2 states that if the decryption / re-encryption cost at a child SAKM agent y is less important than the mutual impact re-keying cost induced by its merge with its parent subgroup, then y can decide to become *active* and that this decision is optimal. Relying on this corollary, SAKM proceeds as follows: periodically, each SAKM agent x computes new estimations of the two parameters λ_x (the mean arrival rate of members at the agent's subgroup) and μ_x (where $\frac{1}{\mu_x}$ is the mean membership duration of the members of the agent's subgroup). x sends these parameters (λ_x, μ_x) to its child SAKM agents (y). Each child SAKM agent (y) compares then the two costs: $\alpha\tau(P_y, r)$ and $E\left[M_{\{x,y\}}\right]$. If $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right]$, then y becomes *active* (corollary 2). If $\alpha\tau(P_y, r) > E\left[M_{\{x,y\}}\right]$, then y becomes *passive*. The decision to become *passive* may be not optimal. In fact, each child decides to become *passive* without taking into consideration the mutual impact re-keying cost due to the other children that might merge with their parent subgroup. However, if x's children (y) verify the inequation $\alpha\tau(P_y, r) > E\left[M_{\{x,y\}}\right]$, it means that the mutual impact between x and y is upper bounded by the constant: $\alpha\tau(P_y, r)$, and hence we can merge them into the same cluster to use the same x's keying material even if this may not be the best configuration (cf. simulation results in section 4.5).

Each SAKM agent y holds two Traffic Encryption Keys (TEKs): TEK_y used in its own subgroup and TEK_x used in its parent subgroup (x). Note that if y is *passive* then $TEK_y = TEK_x$. If y is *active* then, it decrypts received messages using TEK_x and re-encrypts them toward its own subgroup using TEK_y.

Five types of messages are involved in the protocol:

1. *NEW_TEK_RQ*: this type of message is sent by a *passive* agent when a membership change occurs in its subgroup. This message specifies the membership change type (join or leave) and it is sent to the cluster's root agent which is responsible for delivering TEKs for the cluster.

2. *IM_ACTIVE*: this type of message is sent by an agent when it becomes *active*. The message is sent to the cluster's root agent. Upon receiving the message, this latter distributes a new TEK to be used by the remaining subgroups in the cluster.

3. *NEW_TEK*: this message type is used by an *active* agent to distribute a new TEK to its cluster. It specifies the new TEK and the agent's identity which is necessary for the *passive* agents to request new TEKs whenever membership changes occur in their subgroups. It specifies also the membership change type which caused delivering the new TEK, because the distribution scheme of the new TEK depends on the membership change type (join or leave). The keys used to encrypt the new TEK depend also on the membership change type (cf. subsection 4.2.3).

4. *JOIN_LEAVE*: these messages are sent by the members to join or leave the virtual multicast group. An expelling of a member by an agent is considered as receiving a LEAVE message. This type of message specifies the membership change type along with the required member authentication information.

5. *NEW_DYN_INF*: this message is sent by SAKM agents to refresh their dynamism parameters estimation. Each agent sends this message periodically to its child agents. It specifies the new estimations of the two parameters λ and μ. Upon receiving this message, the child agents decide whether to become *passive* or *active*.

Table 4.4 shows the notation and primitives that are used to write algorithms in the remaining of this chapter.

4.4.2 Merge / Split Protocol

In this phase of SAKM, the hierarchy is partitioned into clusters that use the same keying material. Periodically (let say after each θ time units), each SAKM agent x sends the new estimations of λ_x and μ_x to its children y. x signs the message with its private key k_x^{-1} to ensure non-repudiation, the authenticity and the integrity of the sent parameters. Upon receiving λ_x and μ_x (the parent's

$\{m\}_k$	means the message m is encrypted with k using a symmetric encryption algorithm such as DES [109] or AES [111].
$\langle m \rangle_{k^{-1}}$	means m is signed with the private key k^{-1} using a signing algorithm such as RSA [117] or DSS [110].
verify(m)	is a primitive that verifies whether the m's signature is correct.
authorized-(request)	is a primitive that authenticates the joining member and verifies its access rights.
tek(request)	is a primitive that extracts the encrypted TEK from the message request.
type(request)	is a primitive that returns the type of the requested membership change (join or leave).
re-key(type)	is a primitive that assures a re-key process depending on the adopted re-key strategy and the type of membership change (cf. subsection 4.2.3).
mutual-Impact-re-key(type)	is a primitive that assures re-keying (according to the membership change type) due to a mutual impact (cf. subsection 4.2.3).
$a \rightarrow b : m$	means a sends m to b.
SUBG_ADD	Multicast address of the subgroup.
CLUSTER_ROOT_ADD	Address of the cluster root.

Table 4.4: Notation and primitives

parameters), each child SAKM agent y compares $\alpha\tau(P_y, r)$ to $E\left[M_{\{x,y\}}\right]$ and takes the decision to become *active* or *passive* according to the results of that comparison. Figure 4.6 summarizes this phase.

Let x and y be SAKM agents with x parent of y.

Agent x
$x \rightarrow y :< NEW\_DYN\_INF, \lambda_x, \mu_x, timeStamp >_{k_x^{-1}}$ /* periodically */

Agent y
If (state=PASSIVE and $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right]$) **then**
1. $y \rightarrow SUBG\_ADD :< NEW\_TEK, \{newTEK\}_{oldTEK}, JOIN, myID, timeStamp >_{k_y^{-1}}$;
2. $y \rightarrow CLUSTER\_ROOT\_ADD :< IM\_ACTIVE, timeStamp >_{k_y^{-1}}$;
3. state:=ACTIVE;
end if
if(state=ACTIVE and $\alpha\tau(P_y, r) > E\left[M_{\{x,y\}}\right]$) **then**
4. $y \rightarrow SUBG\_ADD :< NEW\_TEK, \{parentTEK\}_{oldTEK}, JOIN, clusterRoot, timeStamp >_{k_y^{-1}}$;
5. state:=PASSIVE;
end if

Figure 4.6: Split / merge protocol

When y becomes *active*, it generates a new TEK_y and multicasts it to the members of its subgroup (line 1). Note that as y's child SAKM agents are members in y's subgroup, they receive the new TEK_y. When a *passive* child SAKM agent receives the new TEK_y , it forwards its distribution to its subgroup and the process continues until all the members in the cluster that will use y's TEK receive the new TEK_y . y informs then the cluster's root agent (t) about the decision to become *active* (line 2) using the "IM_ACTIVE" message. Upon receiving the message, t distributes a new TEK_t to the remaining subgroups in the cluster. This update of TEK_t is compulsory to ensure backward and forward secrecy. In the case where y becomes *passive*, it multicasts its parent's TEK TEK_x to the members of its subgroup (line 4) and changes its state to *passive*. This TEK is

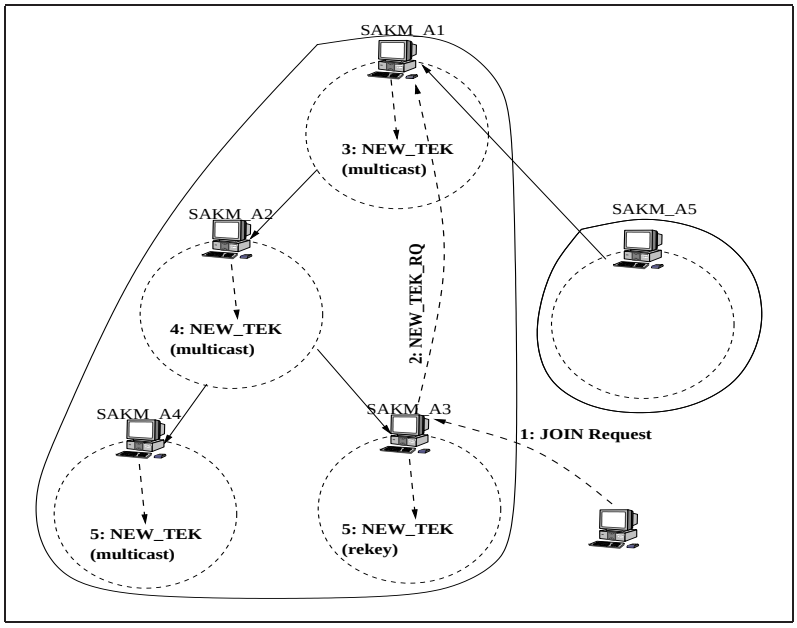


Figure 4.7: Membership change protocol (illustrative example)

forwarded by y's children to all the members in the y's old cluster and informs SAKM agents of the cluster about the new cluster's root.

4.4.3 Membership change protocol

The main idea in this phase is to restrict re-keying to the cluster where occurs the membership change (join or leave). This minimizes the *1 affects n* phenomenon as only the members of the cluster are concerned by the re-key. As the members of a cluster use the cluster's root TEK, when a membership change occurs in a subgroup, the join / leave information is sent to the cluster's root which is responsible for generating and distributing a new TEK for the valid members in the cluster. When a membership change occurs in a subgroup, the SAKM agent responsible for that subgroup reacts as follows: If the agent is *active* (which means that it is a cluster's root), it generates and distributes a new TEK to its subgroup and hence to its cluster (the distribution is forwarded by its child SAKM agents). If the agent is *passive* (an internal agent in a cluster), it sends a request to the cluster's root asking for a new TEK for the cluster ("NEW_TEK_RQ"). Then the cluster's root agent generates and distributes a new TEK for the cluster. All the agents of the cluster distribute the new TEK to their subgroup members according to their re-key strategy and depending on the membership change type (join or leave). In figure 4.7, the numbers show the order of the steps to be taken in order to re-key a cluster following a membership change.

4.4.4 Agent's dynamic behavior

As shown above, the behavior of a SAKM agent depends on its state (*active* or *passive*): this is due to the adaptive and dynamic aspect of the protocol. A SAKM agent can take three states:

1. *active*: in the case the agent is a cluster's root. In this state the agent assures decryption / re-encryption operations, and is responsible for generating a new TEK whenever a membership change occurs in its cluster.

2. *passive*: in the case the agent is a cluster's internal node. In this state, the agent forwards messages without decryption / re-encryption, forwards also up to date TEKs and asks the cluster's root agent new TEKs whenever membership changes occur in its subgroup.

3. *waiting TEK*: this state is introduced to simplify writing the protocol processed by a SAKM agent. An agent is in this state when it is *passive* and waiting an up to date TEK from the cluster's root agent.

The state chart of figure 4.8 depicts the SAKM agent's behavior (operations and actions triggered by events and received messages according to the state of the agent). The numbers in figure 4.8

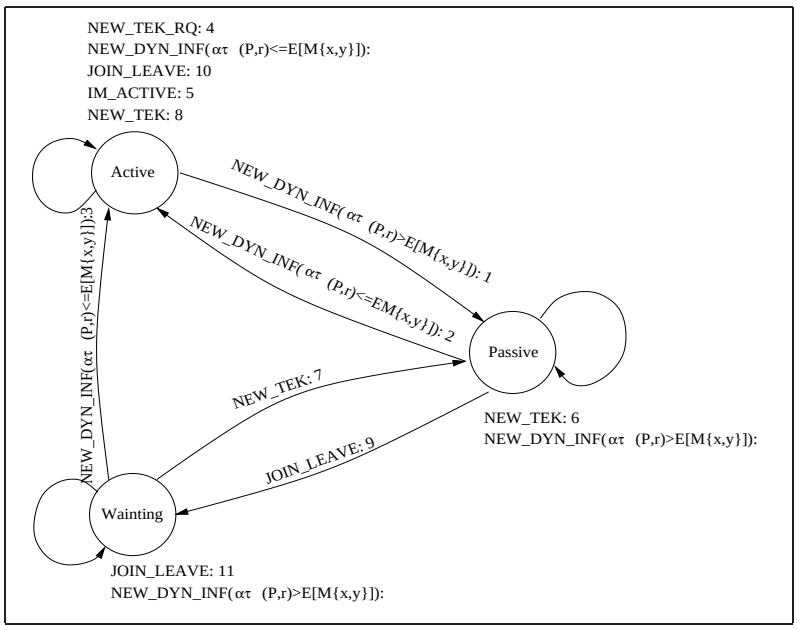


Figure 4.8: A SAKM agent state chart

correspond to the numbered code portions and are commented in the following paragraphs. Each procedure deals with a message type according to the agent's state. We suppose that the procedures are processed by the agent y whose parent is x.

procedure *newDynInf(request)*{
if(state=ACTIVE and $\alpha\tau(P_y, r) > E\left[M_{\{x,y\}}\right](request)$) {
$y \rightarrow SUBG\_ADD :< NEW\_TEK, \{parentTEK\}_{oldTEK}, JOIN, clusterRoot, timeStamp >_{k_y^{-1}}$;
$state := PASSIVE$; } (1)
}
if(state=PASSIVE and $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right](request)$) {
$y \rightarrow SUBG\_ADD :< NEW\_TEK, \{newTEK\}_{oldTEK}, JOIN, myID, timeStamp >_{k_y^{-1}}$;
$y \rightarrow CLUSTER\_ROOT\_ADD :< IM\_ACTIVE, timeStamp >_{k_y^{-1}}$;
$state := ACTIVE$; } (2)
}
if(state=WAITING and $\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right](request)$) {
re-key(type(request));
$y \rightarrow CLUSTER\_ROOT\_ADD :< IM\_ACTIVE, timeStamp >_{k_y^{-1}}$;
$state := ACTIVE$; } (3)
}
}

Figure 4.9: Procedure that deals with NEW_DYN_INF messages

Receipt of *NEW_DYN_INF* message

In actions (1)(fig.4.9), an *active* agent becomes *passive*: it multicasts (to its subgroup) its parent's TEK encrypted with the TEK used in its subgroup (oldTEK). In actions (2)(fig.4.9), a *passive* agent becomes *active*: it generates a new TEK and multicasts it to its subgroup (and hence to the new cluster for which it becomes a root), then it informs the cluster's root about the change in its state, so that the latter updates the TEK for the remaining subgroups in the cluster. If the agent is *waiting* for a new TEK because of a membership change in its subgroup and receives a NEW_DYN_INF message which states that it should become *active* ($\alpha\tau(P_y, r) \leq E\left[M_{\{x,y\}}\right]$), then the agent generates a new TEK and re-key its subgroup according to the adopted strategy and the membership change type. The agent informs then the cluster's root agent about the change of state (actions (3)-fig.4.9).

procedure *newTekRq(request)*{
$mutualImpactre - key(type(request))$; } (4)
}

Figure 4.10: Procedure that deals with NEW_TEK_RQ messages

Receipt of *NEW_TEK_RQ* message

Action (4)(fig.4.10) consists of a mutual impact re-keying which depends on the adopted strategy and the membership change type for which the new TEK is requested (cf. subsection 4.2.3).

procedure *imActive(request)*{
$y \rightarrow SUBG\_ADD :< NEW\_TEK, \{newTEK\}_{oldTEK}, type(request), clusterRoot, timeStamp >_{k_y^{-1}}$; } (5)
}

Figure 4.11: Procedure that deals with IM_ACTIVE messages

Receipt of *IM_ACTIVE* message

When a cluster's root agent receives a IM_ACTIVE message, it generates a new TEK and multicasts it to its subgroup (and hence to the remaining subgroups in the cluster) encrypted with the old TEK (action (5)-fig.4.11).

```
procedure newTek(request){
  if(state = PASSIVE and verify(request)){          } (6)
  currentTEK := tek(request);                       }
  y → SUBG_ADD : request;                           }
  }
  if(state = WAITING){                              } (7)
  re − key(request);                                }
  state := PASSIVE;                                 }
  }
  if(state = ACTIVE){                               } (8)
  parentTEK := tek(request);                        }
  }
}
```

Figure 4.12: Procedure that deals with NEW_TEK messages

Receipt of *NEW_TEK* message

When a *passive* agent receives an up to date TEK, it forwards it to its subgroup (actions (6)-fig.4.12). If the agent is *waiting* for a new TEK, then it makes a re-key of its subgroup and resumes the *passive* state (actions (7)-fig.4.12). Finally, if the agent is a cluster's root, it takes note that its parent updated its TEK (action (8)-fig.4.12).

```
procedure joinLeave(request){
  if(state = PASSIVE and authorized(request)){                                  } (9)
  y → CLUSTER_ROOT_ADD :< NEW_TEK_RQ, type(request), timeStamp >_{k_y^{-1}};    }
  state := WAITING;                                                             }
  }
  if(state = ACTIVE){                                                           } (10)
  re − key(type(request));                                                      }
  }
  if(state = WAITING){                                                          } (11)
  /* as the agent is waiting for a new TEK, it does not send an other          }
  NEW_TEK_RQ, but only takes into consideration the new membership              }
  change when the new TEK is ready */                                          }
  }
}
```

Figure 4.13: Procedure that deals with JOIN_LEAVE messages

Receipt of *JOIN_LEAVE* message

When a membership change occurs in a subgroup whose SAKM agent is *passive*, the agent asks the cluster's root agent a new TEK with a message NEW_TEK_RQ and changes its state to WAITING (action (9)-fig.4.13). However, if the agent is active (a cluster's root) then it generates and

distributes a new TEK according to the adopted re-key strategy and the membership change that occurred in its subgroup (action (10)-fig.4.13).

4.5 Simulation results

In this section, we provide an overview of our simulation model and some of the results we obtained by comparing SAKM with Iolus [87] and the centralized solution GKMP [66]. We selected Iolus [87] as a *TEK per subgroup* approach representative protocol. In this approach, the multicast group is divided into multiple subgroups, in a static manner, with independent TEKs and thus it suffers from the high number of decryption / re-encryption operations. We selected GKMP [66] as a representative protocol of the *common TEK* approach. In this approach, members share a same TEK and thus suffer from the *1-affects-n* phenomenon. We study the *1 affects n* behavior of each simulated protocol and the number of decryption / re-encryption operations required for the communication.

4.5.1 Simulation model

In our simulation, we use a virtual SAKM multicast group made up of five multicast groups organized as shown in figure 4.7. We suppose that the group is composed of 100 dynamic members in the average. In order to show the ability of SAKM to cope with different application requirements, we make the simulation for three types of applications characterized by their requirement in term of synchronization between the source and receivers (cf. table 4.5).

Application	*Synchronization*
T1	Low synchronization required
T2	High synchronization required (small latencies are allowed)
T3	Real-time (no latency allowed)

Table 4.5: Three application types

T1 corresponds to applications that do not need real-time data transmission such as replicating distributed data bases, software updating or broadcasting stock quotes. T2 corresponds to applications that need some synchronization between the source and receivers but may tolerate small latencies, some TV shows on the Internet could be classified in this category. T3 represents applications that require real-time data transmission such as video-conferencing. Our multicast sessions are generated using the models presented by Almeroth et al. in [2][3]. These models suggest that, for some multicast sessions, the arrival of members follows a Poisson process and the membership duration follows an Exponential distribution. These models are deduced from real multicast sessions observed on the Mbone. In the carried out simulation, we consider a session of 3 hours, an inter-arrival between members of 20 seconds and an average membership duration of 30 minutes. We suppose that the distribution of arrival members at the different subgroups is not uniform and changes over time.

4.5.2 Split / merge criteria

Experiments [35] show that with a Celeron 850 MHz processor, to encrypt or decrypt a message of 1 M Bytes we have the results of table 4.6

Algorithm	*Encryption time*
DES	$78ms$
AES	$33ms$
IDEA	$88ms$

Table 4.6: Required time to encrypt a message of 1 M Bytes

If we consider a flow with a rate r ($2^{-3}Mbytes/s$ or $16kbps$ of encrypted data), and we use DES to assure its secrecy, then per each r Mbytes of received data, we would have an overhead of $2.r.DES_t$ (2 for decryption and re-encryption, $DES_t = 78ms$ in our case) seconds for decryption / re-encryption. Thus the decryption / re-encryption overhead becomes

$$\alpha\tau(P_i, r) = 2.\alpha.r.DES_t$$

We will show that α plays a key role in controlling the behavior of SAKM regarding synchronization requirements of the application. Consider two adjacent subgroups of a SAKM hierarchy i and j with i parent of j. Suppose that both SAKM agents of subgroups i and j use the hierarchical key graph scheme for re-keying and that the graph is a full binary tree. Thus the mutual impact re-keying cost would be

$$E\left[M_{\{i,j\}}\right] = 3(\lambda_i + \lambda_j)$$

(cf. table 4.2). Hence, for j to take a decision about its state, it has to compare between the two costs $2.\alpha.r.DES_t$ and $E\left[M_{\{i,j\}}\right] = 3(\lambda_i + \lambda_j)$ after each θ units of time ($15mins$ in our simulation).

4.5.3 Simulation results and discussion

In a first stage, we consider a T1 application that do not require much synchronization. SAKM aims to minimize decryption / re-encryption overhead as well as the *1 affects n* phenomenon according to the members' dynamism. Figures 4.14,4.15,4.16 show the results obtained with a T1 application: figure 4.14 measures the number of decryption / re-encryption operations which corresponds to the number of clusters in SAKM and to the number of subgroups in Iolus (5 in our case). Figure 4.15 measures the number of affected members: with the centralized solution all the members are affected, with Iolus only the members of a subgroup are affected, that is why the results of Iolus are much smaller, SAKM makes a trade off between decryption / re-encryption overhead and *1 affects n*. Figure 4.16 gives the same results of figure 4.15 in the average which means that it divides the number of affected members by the number of *active* clusters. As T1 tolerates latencies, we give to the weight α a small value (4 in our case), so that SAKM creates as much clusters as it needs to attenuate *1 affects n* and to minimize decryption / re-encryption

overhead compared to Iolus which makes it systematically at each Iolus agent. At a first sight, we remark that even if SAKM makes only three decryption / re-encryption operations in the average (cf. figure 4.14), it maintains as good performances as those of Iolus (cf. figures 4.15,4.16).

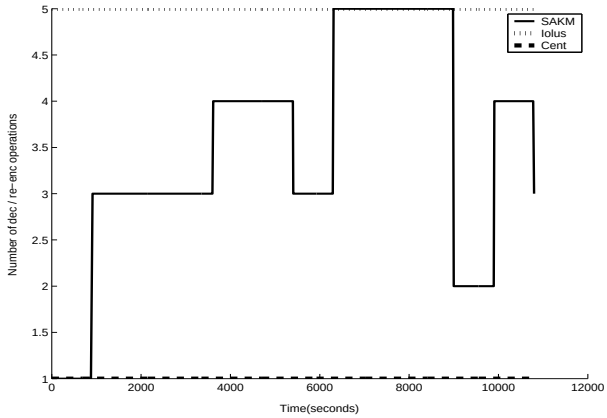


Figure 4.14: Dec / Re-enc operations (T1)

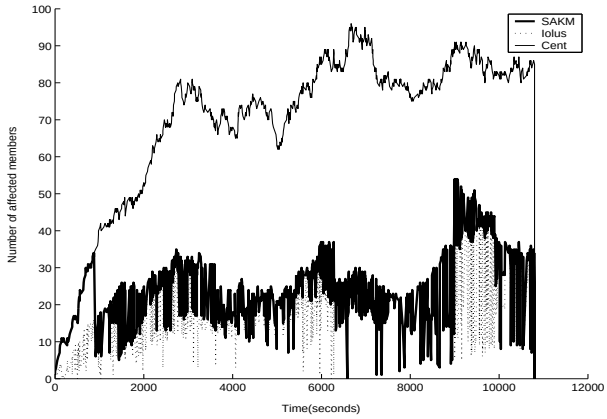


Figure 4.15: *One-affects-n* phenomenon (T1)

SAKM starts with a centralized behavior (a single cluster) ($0 < t < 700seconds$). As the group size grows, the group dynamism increases and thus SAKM creates new clusters following the split / merge process; at $t = 700s$ SAKM creates only 3 clusters (cf. figure 4.14) and reaches with that Iolus performances regarding *1 affects n* attenuation (cf. figures 4.15, 4.16), which means that SAKM saves decryption / re-encryption overhead as well as re-keying messages overhead, which is not the case with Iolus. At each time the group dynamism reaches a certain degree, SAKM creates a new cluster and hence the number of decryption / re-encryption operations increases by one and in the counter part *1 affects n* is much more attenuated (see for example: figure 4.14 and figure 4.15 at $3800s < t < 5500s$ and at $6500s < t < 9000s$). Whenever decryption / re-encryption cost $2.\alpha.r.DES_t$ exceeds the mutual impact re-keying cost, SAKM destroys as much clusters to reach a better whole partition cost (cf. figure 4.14 and 4.15 at $t = 5500s$ and $t = 9000s$). In this way, SAKM assures a trade off between decryption / re-encryption cost and re-keying cost so that

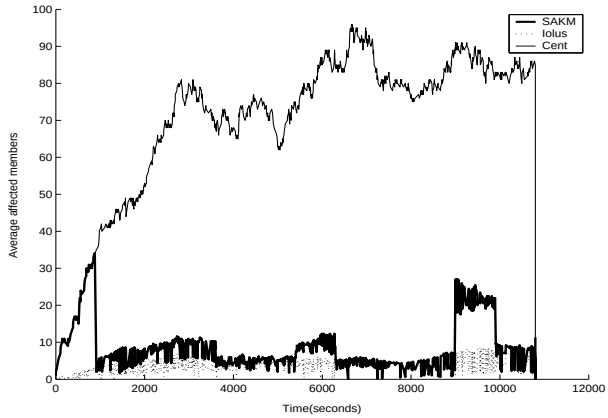


Figure 4.16: Average *one-affects-n* phenomenon (T1)

the whole cost is minimized.

In a second stage, we consider a T2 application that requires high synchronization between the source and receivers. Iolus do not fit with this kind of applications since decryption / re-encryption operations would introduce latencies that are not desirable. With SAKM, we give to the factor α a value (8 in our case) that prevent it from creating a lot of clusters except for situations where the membership changes in a sharply way and it would be better to limit the re-key to the subgroup(s) where the membership changes. Such a situation is very rare and do not affect performances of T2 that tolerates some latencies (it is not a serious problem when we receive a slightly slow sequence for a while when seeing a movie on the Internet). In figure 4.17 we remark that the state of SAKM agents does not change frequently, and that it happens only when the membership changes in a sharp way (cf. figures 4.17, 4.18 at $4500s < t < 8000s$) where SAKM creates three clusters because of a sharply change in the membership change.

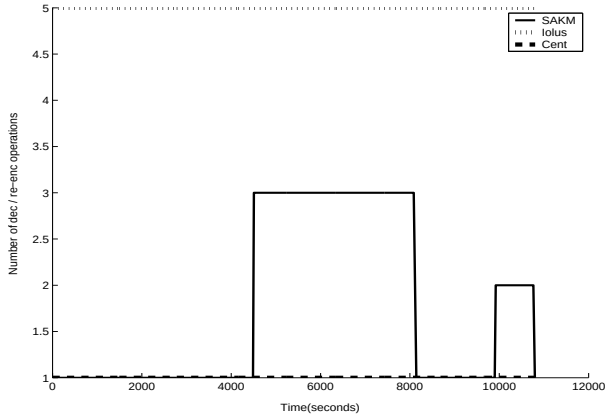


Figure 4.17: Dec / Re-enc overhead (T2)

Finally we consider a T3 application, where it is out of question to do decryption / re-encryption of the data which should reach receivers in real-time. Iolus does not support this kind of requirement.

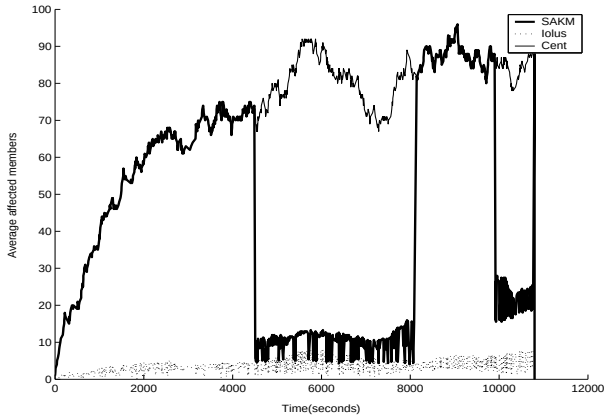


Figure 4.18: Average *one-affects-n* phenomenon (T2)

With SAKM it suffices to put the factor α to infinite theoretically (16 in our case) to prevent SAKM from creating clusters. And hence SAKM becomes typically a centralized solution without intermediaries (cf. figures 4.19, 4.20).

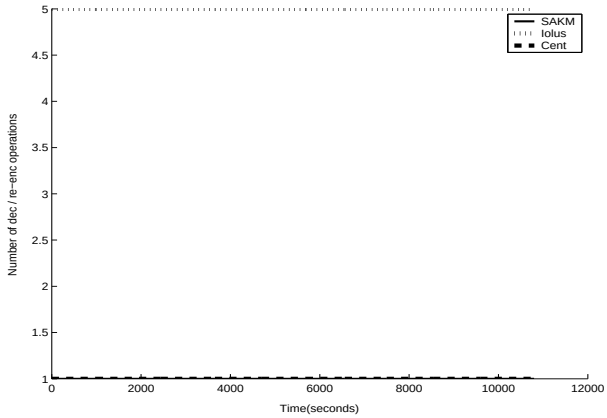


Figure 4.19: Dec / Re-enc overhead (T3)

4.6 Conclusion

Secrecy is an urgent requirement for multicasting in order to ensure a safe and large deployment for confidential group communications. Key management protocols play a key role in the whole secure multicast architecture. In real multicast sessions, members can join and leave the group dynamically during the whole session. This dynamism affects considerably the performances of the key management protocol. Most proposed solutions in the literature do not take this parameter into consideration and so suffer either from the *1-affects-n* phenomenon or from the important data translation overhead. In this chapter, we considered a special class of group key management which organizes the multicast group into subgroups with independent traffic encryption keys so

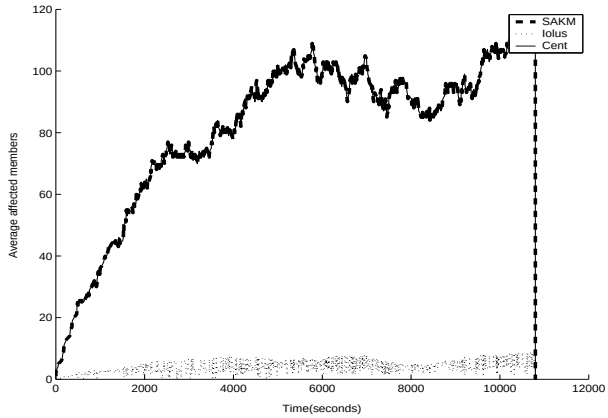


Figure 4.20: Average *one-affects-n* phenomenon (T3)

that the *1-affects-n* phenomenon and hence the re-keying overhead are minimized. In this kind of architectures, multicast messages should be translated at the boundaries of subgroups. The translation operations create a new overhead which may be disastrous for some kind of applications that require a real-time or highly synchronized data transmission. We showed that it is possible to make a trade off between the two overheads (data translation and re-keying overheads) by making an adaptive clustering of subgroups. We proposed a heuristic to approach the optimal configuration of this clustering and the simulation results show that it is a good approach compared to two other approaches from the literature. Then, we proposed a new *dynamism aware* key management protocol called: SAKM. Simulations show the efficiency of SAKM because of its ability to tune and adapt the dimensions of key management areas, and thereby to achieve the adequate performance trade-off between data transformation and re-keying overheads.

Part II

Data Origin Authentication in Group Communication

Chapter 5

Definitions and Requirements

IN this chapter, we recall how to use some cryptographic mechanisms to ensure data integrity, data origin authentication and non-repudiation. Then we introduce issues and requirements of data origin authentication in the context of group communication.

5.1 Data integrity

Data integrity is the property that data has not been changed, destroyed, or lost in unauthorized or accidental manner [128]. *Cryptographic hash functions* are typically used to ensure *data integrity* [83].

Definition 2 ***A hash function*** *is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values [83].*

We denote the hash-value of a message m by $h(m)$. Cryptographic *hash functions* have the following properties [122, 83, 71, 128]:

1. Given m, it is easy to compute $h(m)$.
2. Given d, it is hard to compute m such that $h(m) = d$.
3. Given m, it is hard to find another message, m', such that $h(m) = h(m')$.

A *hash-value* is also called *message digest*, *hash-result*, or simply: *a hash*.

Example:[1] Suppose that you want to save a large digital document (a program or a database) from alterations that may be caused by viruses or accidental mis-uses. A straightforward solution would be to keep a copy of the digital document on some tamper-proof backing store and periodically compare it to the active version. With a *cryptographic hash function*, you can save storage: you simply save the message digest of the document on the tamper-proof backing store (which because the hash is small could be a piece of paper or a floppy disk) (cf. figure 5.1, steps 1 and 2). Then, periodically, you re-calculate the message digest of the document (cf. figure 5.1, step 3)

[1] This example is cited by many authors such as Kaufman et al. in [71] and Menezes et al. in [83]

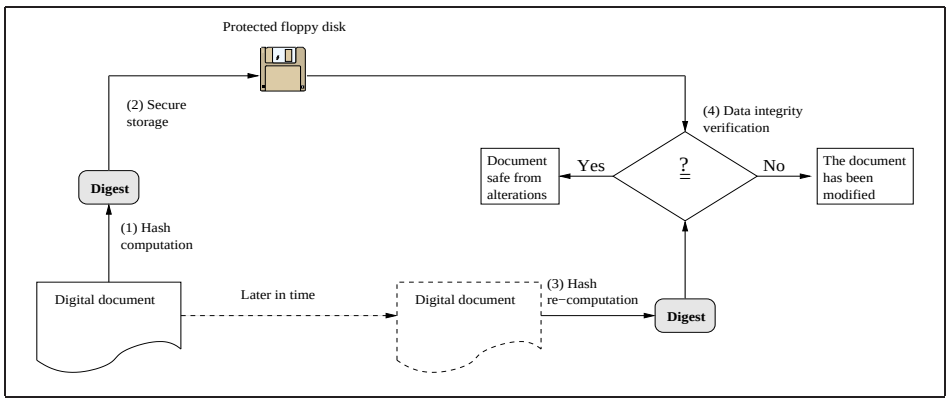


Figure 5.1: Assuring data integrity using Message Digests

Hash Algorithm	Digest Size (bits)	Computation Speed (MBytes / second)
MD5	128	204.55
SHA-1	160	72.60

Table 5.1: Measurements of some hash algorithms

and compare it to the original message digest (cf. figure 5.1, step 4). If the message digest has not changed, you can be confident none of the data has. Examples of hash functions are: MD2 (Message Digest 2)[70], MD5 [116], SHA-1 (Secure Hash Algorithm 1)[49]. Table 5.1 gives some measurements for usually used hash algorithms. These measurements are from a benchmark due to Dai [35]. The author used Pentium 4 2.1 GHz processor under Windows XP SP 1.

5.2 Data origin authentication

Data origin authentication is the corroboration that the source of data received is as claimed [128]. *Message Authentication Codes (MACs)* is a cryptographic mechanism that can be used to assure data origin authentication and data integrity at the same time.

Definition 3 ***A Message Authentication Code (MAC) algorithm*** *is a family of functions h_k parameterized by a secret key k, with the following properties [83]:*

1. *Given a key k and an input m, $h_k(m)$ is easy to compute.*
2. *h_k maps an input m of an arbitrary finite bit length to an output $h_k(m)$ of fixed bit length. Furthermore, given a description of the function family h, for every fixed allowable value of k (unknown to an adversary), the following property holds:*
3. *Given zero or more pairs $(m_i, h_k(m_i))$, it is computationally infeasible to compute any pair $(m, h_k(m))$ for any new input m.*

MAC Algorithm	Digest Size (bits)	Computation Speed (MBytes / second)
HMAC/MD5	128	215.76

Table 5.2: Measurements of HMAC

A MAC can also be seen as *"a cryptographic hash in which the mapping to a hash result is varied by a second input parameter that is a cryptographic key"* [128]. Thus, the point of a MAC is to send something that only someone knowing the secret key can compute and verify. For example, a MAC can be constructed by concatenating a shared secret K_{AB} with the message m, and use $H(m|K_{AB})$ as the MAC (where H is a hash function)[71].

Thus, to assure data origin authentication, a sender A and a receiver B have to share a secret key K_{AB}. Then the sender computes the digest ($MAC(K_{AB}, m)$) corresponding to the message (m), to be sent, using the secret key (K_{AB})(cf. figure 5.2 step 1). Upon receiving the message as well as the digest, the receiver verifies the origin of the received message as follows: it recalculates the digest of the received message using the secret key K_{AB} (fig. 5.1 step 3) and compares it to the received digest (fig. 5.1 step 4). If the two digests are equal, the message is said to be authentic (has not been altered) and originates from the sender A since only A and B know the secret K_{AB}. Otherwise, the received message has been altered or fabricated by a sender who is not A. An

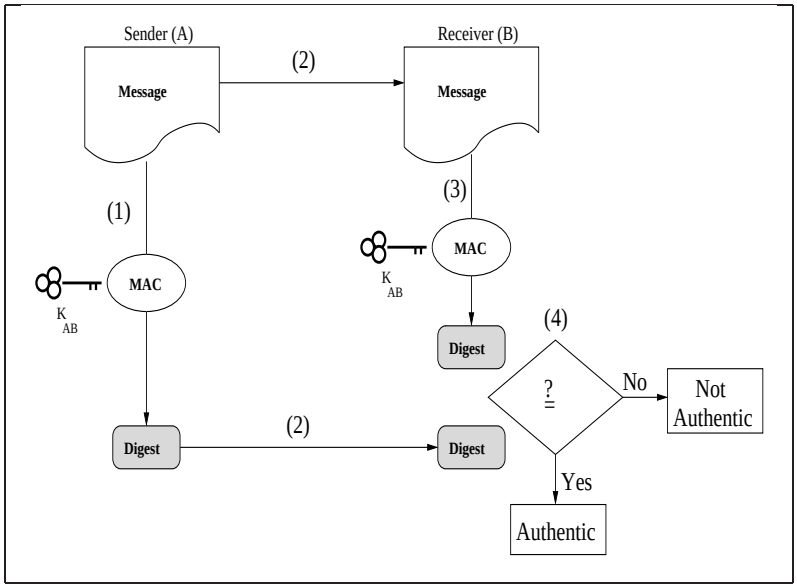


Figure 5.2: Assuring data origin authentication using MACs

example of MAC is: HMAC [76]. Table 5.2 gives some measurements for HMAC.

5.3 Non-repudiation with proof of origin

Non-repudiation with proof of origin provides the recipient of data with evidence that proves the origin of the data, and thus protects the recipient against an attempt by the originator to falsely deny sending the data [128].

Note that a MAC cannot be used as a proof (to a third party) that a message originates from a specific entity. In fact, let us consider that a sender A and a receiver B share a secret key K_{AB}. If A denies having sent a message m, the receiver B cannot use the received MAC of m as a proof of m's origin, because A would then say that B might have created the m's MAC himself!. Thus, asymmetric cryptography is the basic answer for non-repudiation. With asymmetric cryptography, the piece of information sent with the message as a proof of integrity and data origin is computed using a private key held only by the sender and is verified by the receiver using the public key that corresponds to the private key. Hence, since only the sender can compute the piece of information, this latter can be used as a proof of origin to a third party and hence non-repudiation is assured. This cryptographic mechanism is called *Digital Signature*.

To *sign* a message, a sender generates a pair of private/public keys using some asymmetric cryptographic system. The sender keeps the private key secret and publishes the public key. Then the sender calculates the digest of the message to be sent using any hash function (cf. figure 5.3 step 1). The digest is then cryptographically transformed using the private key (fig. 5.3 step 2). The result of this transformation is called: the *digital signature* of the message. Upon receiving the message and the signature, the receiver verifies the signature using the *public key* as follows: first, the receiver recalculates the digest of the received message (fig. 5.3 step 4). Then, the receiver verifies the received signature using the public key (fig. 5.3 step 5). If the signature is valid then the message as well as its origin are authentic and non-repudiation is guaranteed. Otherwise the message is rejected. Examples of digital signature schemes are: RSA [117], DSA [110]. Table 5.3

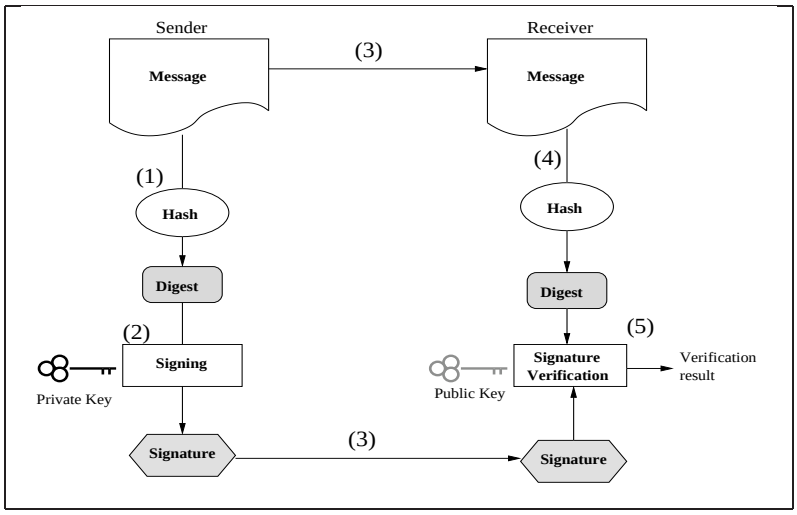


Figure 5.3: Assuring non-repudiation using Digital Signatures

Digital Signature	Signing Speed (ops / second)	Verification Speed (ops / second)	Public Key Size (bits)
RSA-1024	215	5263	1024
DSA-1024	467	420	1024

Table 5.3: Measurements of some digital signature systems

gives some measurements for usually used digital signature systems.

5.3.1 Certification

To verify a signature, a receiver needs to be assured that the public key used in verifying a signature corresponds to the private key of the real sender of the signed message and not generated by an intruder who tries to impersonate the real sender. The electronic document that assures this matching is called: *public-key certificate*.

Definition 4 ***A public-key certificate*** *is a digital certificate that binds a system entity's identity to a public key value, and possibly to additional data items; a digitally-signed data structure that attests to the ownership of a public key [128].*

Definition 5 ***Certification*** *is the process of vouching for the ownership of a public key by issuing a public-key certificate that binds the key to the name of the entity that possesses the matching private key [128].*

A *certificate* is digitally signed by a *Certification Authority (CA)* which is *trusted* by receivers and whose public key is known by receivers in a secure way. Thus, to publish a public key, a sender should issue a signed certificate of its public key to receivers. The enclosed public key is then used by receivers to verify the digital signatures generated by the sender whose identity is also enclosed in the same certificate.

In the rest of the manuscript, we suppose that the public keys used to verify digital signatures are certified.

5.3.2 One-time signing

Conventional digital signature mechanisms such as RSA and DSA are very *computationally expensive*. *One-time signing* is a *fast* alternative scheme, with the price of a *weaker* security that limits the use of a pair of *one-time private/public* keys to a single message. The general idea of a one-time signature scheme is that the private key is used as the input to a sequence of *One Way Function* evaluations which result in a sequence of intermediate results and finally in the public key. The "one-wayness" of the functions implies that it is infeasible to compute the private key, or any intermediate result of the computation, from the public key. A signature for a given message

consists of a subset of the intermediate results of this computation, where the message to be signed determines which particular subset is revealed as the corresponding signature [13, 14]. To verify a one-time signature, a receiver applies a subset of the one way evaluations to the one-time signature. If the result of these evaluations is equal to the public key, then the one-time signature is valid. A one-time signature scheme allows the signature of only a single message using a given pair of private / public keys. One advantage of such a scheme is that it is generally quite fast. However, it is known that the produced one-time signature is quite large. Besides, since a pair of one-time (private / public) keys can be used to sign only a single message, the sender is required to issue a new public-key certificate each time it changes this pair of keys. This very frequent need for new public keys may induce high computation and bandwidth overheads. One-time signature scheme was first introduced by Lamport [79] and more efficient schemes have been proposed since then [84, 85, 14, 102, 88, 115].

The expression *k-time signature* is also used to designate a signature scheme whose pair of private/public keys can be used with k messages at most. In what follows, if we omit to specify that a signature scheme is k-time, then we mean a conventional digital signature scheme, like RSA.

5.4 Multicast Data Origin Authentication Issues and Requirements

In this section, we state the problem of multicast data origin authentication and highlight the issues and challenges that encounter this security service. Consider a *sender* that *streams* data to a set of *receivers* in a multicast session. We consider a *stream* as an *infinite* sequence of packets that are sent successively. Receivers of the multicast session are *not trusted*. The sender *authenticates* a multicast message using some *authentication procedure* which generates the *authentication information* associated to the message. The message along with its authentication information are multicast to receivers. A receiver *verifies* the authentication information using some *verification procedure*. Figure 5.4 summarizes multicast data origin authentication requirements from four points of view:

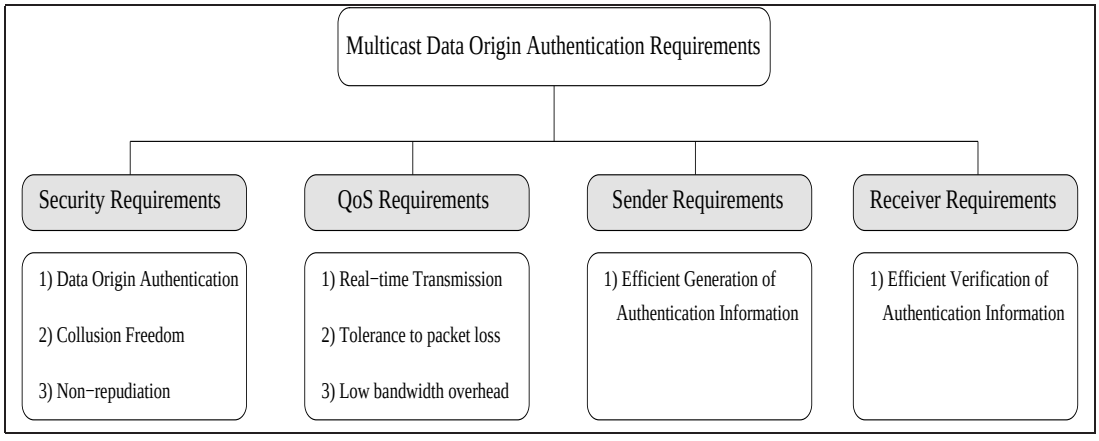


Figure 5.4: Multicast Data Origin Authentication Requirements

Security requirements

1. Receivers require the ability to verify the origin of received data. This requirement is called *data origin authentication*.
2. Any subset of receivers must be unable to collude in order to impersonate the sender of data. This requirement is called *collusion freedom*.
3. The sender must be unable to *deny* having sent data to the multicast session. This requirement is called *non-repudiation*.

QoS requirements

1. The authentication information must induce a *low bandwidth overhead*.
2. Since most of multicast media-streaming applications use unreliable packet delivery, multicast data origin authentication mechanisms must be *robust against packet loss*.
3. Since most of multicast media-streaming applications require *real-time transmission*, multicast data origin authentication must not induce latencies at the sender before authenticating stream packets, nor at receivers before verifying received packets authenticity.

Other requirements

1. The generation of the authentication information must induce *low computation and/or memory overheads*, so that the multicast data origin authentication scheme can be implemented with resource constrained devices.
2. The verification of the authentication information must induce *low computation and/or memory overheads*, so that the multicast data origin authentication scheme can be implemented with resource constrained devices.

In the following chapter, we will see that none of the proposed solutions satisfies all of these requirements, but each proposed approach tries to make the best trade-off from a specific point of view.

5.5 The bursty packet loss model

Packet loss modeling occupies an important place in the literature relating to data origin authentication, because most of multimedia group communication applications do not use reliable transport layer and hence authors need to model this phenomenon in order to evaluate the performance of proposed solutions.

Many studies show that packet loss is correlated, which means that the probability of loss is much higher if the previous packet is lost. Paxson shows in [100] that packet loss is correlated and that the length of losses exhibit infinite variance. Borella et al. found that the average length of loss bursts is about 7 packets [16]. Yanik et al. show that a k-state Markov chain can model Internet packet loss patterns [142]. For our simulation purposes, the two-state Markov chain model is sufficient, since it can correctly model simple patterns of bursty packet loss [142]. Figure 5.5 shows the two-state Markov chain used in our simulations and whose transition probabilities can easily be determined using the average burst length and the packet loss ratio in the network.

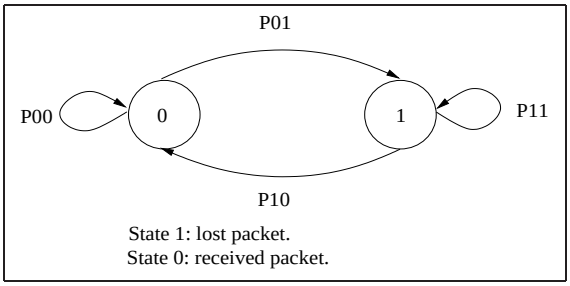


Figure 5.5: Two-state Markov chain to simulate bursty packet loss

5.6 Group Authentication vs. Data Origin Authentication

Even though a multitude data origin authentication mechanisms have existed so far, data origin authentication in multi-party communications remains a challenging problem in terms of scalability, efficiency and performance. In fact, hashes [70, 116, 49], MACs [76] and digital signatures [110, 117] are the cryptographic answers to data integrity, data origin authentication and non-repudiation in data transmission. However, these mechanisms have been designed typically for point-to-point transmissions, and using them in multicasting yields inefficient and non-adequate solutions. This non-suitability of existing authentication mechanisms is mainly due to the number of group members which may be high in multi-party applications, and to the type of transmitted data which consists generally in continuous streaming of multicast messages with real-time transmission. We distinguish between two types of authentication in group communication [64]:

- **Group authentication**: consists of assuring that the received multicast messages by group members originate from a valid group member (no matter its identity).
- **Data origin authentication**: consists of assuring that the received multicast messages by group members originate from a source having a specific identity.

In order to assure *group authentication*, generally group members use a shared key. This key is commonly called: *group key*. Applying a MAC to a message with the group key assures that the message originates from a valid group member, since only valid group members share the group

key. Hence, the group authentication problem is reduced to the *group key management* and mainly to its scalability to large groups [114, 64, 69, 28]. In contrast, *multicast data origin authentication* is more complicated because the *group key* which is known by all group members cannot be used to identify a specific sender. Therefore, it is required to introduce *asymmetry* in the generation process of the authentication information, so that receivers can verify it without being able to generate it on behalf of the legitimate sender.

5.7 Conclusion

Many applications, such as broadcasting stock quotes and video-conferencing require data origin authentication of the received multicast traffic. Multicast data origin authentication must take into consideration the scalability and the efficiency of the underlying cryptographic schemes and mechanisms, because multicast groups can be very large and the exchanged data is likely to be important in volume (streaming). Besides, multicast data authentication must be robust enough against packet loss because most of multicast multimedia applications do not use reliable packet delivery. In the following chapter, we will review different multicast data origin authentication approaches proposed in the literature.

Chapter 6

A Taxonomy of Multicast Data Origin Authentication

MULTICAST data origin authentication has matured over the last twenty years and many solutions have been proposed to solve this challenging problem. In this chapter, we review and classify recent works dealing with the data origin authentication problem in group communication and we discuss their underlying concepts. We classify existing multicast data origin authentication protocols depending on the security objective in a first stage. Thus we distinguish between two major classes of existing protocols: those that assure data origin authentication and non-repudiation, and those that assure data origin authentication but not non-repudiation. Then we refine the classification according to the common underlying solution concept (cf. figure 6.1).

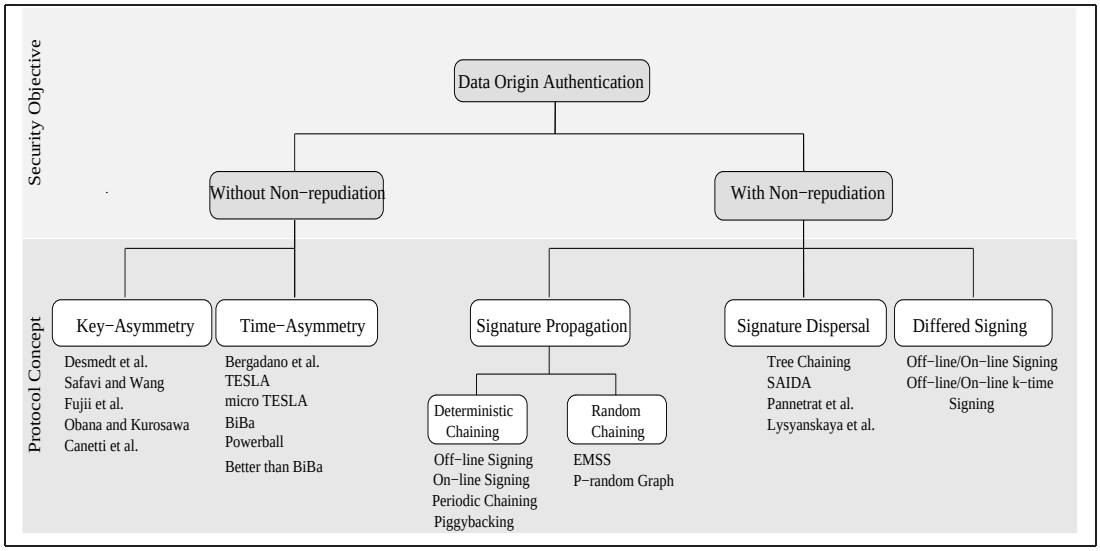


Figure 6.1: Classification of data origin authentication protocols

We give more detailed descriptions of the existing protocols and valuable comparisons with respect to some relevant performance criteria, as well as fruitful discussions in our survey [25].

6.1 Multicast Data Origin Authentication

To assure data origin authentication for a given message, a receiver has to share a secret with the sender. This secret is involved in computing the *authenticator* of the message, and it is used to verify this authenticator by the receiver. Data origin authentication is guaranteed because only the sender and the receiver know the secret. In multicast data origin authentication, a straightforward solution is to make the members of the multicast group share a secret key with the sender. To authenticate a multicast message, the sender computes the MAC of the message using this key and multicasts it along with the message. Upon receiving the message and its MAC, receivers verify data origin authentication by verifying the received MAC using the secret key. The problem with this naive solution is that all group members know the secret key and hence everyone can impersonate the valid sender and forges MACs of messages on the sender's behalf. Therefore, to assure multicast data origin authentication, the authentication information must be *asymmetric*. We mean by *asymmetric* that receivers can verify authentication information but cannot generate it.

Existing solutions considered two major approaches to introduce *asymmetry* in authentication information. In what follows, we discuss and illustrate each sub-category with some relevant solutions from the literature.

6.1.1 Secret-Information Asymmetry

In this approach, the sender uses a set of secrets to authenticate a multicast message and gives to each receiver a partial view of the used secrets that allow him only to verify authenticity of received messages without being able to generate valid authentication information for messages. We distinguish between two sub-categories depending on the desired level of security. Figure 6.2 summarizes this taxonomy refinement.

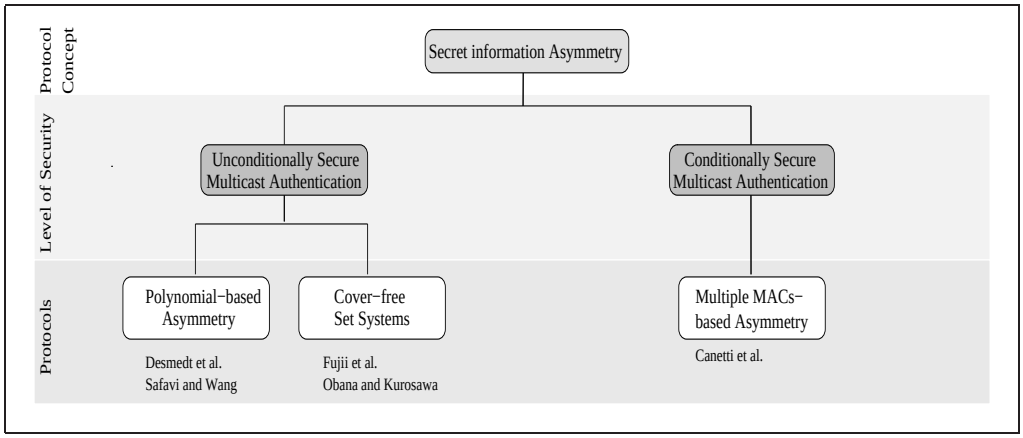


Figure 6.2: Classification of data origin authentication protocols based on *Secret Information Asymmetry*

Unconditionally secure authentication

In this approach it is imperative that the security of the authentication scheme is not based on unproven assumptions. Thus, absolute security is guaranteed by making sure that attackers cannot get enough information which enables attacks. Protocols of this approach rely on information theoretic strength.

DESMEDT ET AL. PROTOCOL: Desmedt et al. [41] introduced k out of n Multi-Receiver Authentication schemes ((k, n) MRA) as follows:

Definition 6 *An authentication scheme is a k out of n multi-receiver authentication scheme, when any k out of n receivers cannot commit a substitution or impersonation attacks on even a single receiver.*

A *substitution* attack is committed when a receiver accepts a message which has been altered by the attacker in course of transmission, and an *impersonation* attack succeeds when a receiver accepts a message from an attacker on behalf of a legitimate sender. In other words, in a k out of n authentication scheme, the largest coalition of cheating receivers can have $k - 1$ members out of n receivers.

Desmedt et al. [41] proposed a polynomial based scheme to assure a k out of n multicast authentication. The main idea of the proposed scheme is that the source generates a polynomial $A_M(x)$ of degree k, using the message M to be sent. Then the source sends to each receiver a share of the polynomial, in such a way that to forge an authenticator of a message, it is required to use at least k shares of the polynomial to reconstruct it using *interpolation technique*. Since the largest coalition of cheating receivers can have only $k-1$ members, the security of the system holds. The following steps describe the proposed protocol:

1. The sender chooses a large prime p where p is equal or greater than the number of possible messages. All the following operations are done in the finite field Z_p (integers modulo p).
2. For each message M, the sender creates two polynomials $P_0(x)$ and $P'(x)$ of degree k.
3. The sender privately transmits the shares $P_0(i)$ and $P'(i)$ to each receiver (i).
4. The sender generates the authenticator (polynomial) of the message M: $A_M(x) = P_0(x) + M \times P'(x)$ and multicasts it to receivers.
5. The receivers verify the authenticator by testing if $A_M(i) = P_0(i) + M \times P'(i)$.

Authors proved in [41] that this protocol is a k out of n *unconditionally secure authentication scheme*.

This protocol has the merit to be a pioneer to introduce multicast data origin authentication using unconditional security. It is clear that this scheme tolerates packet loss, since each packet carries

its own authentication information and hence can be verified independently from other packets as soon as the packet is received. Desmedt et al. showed that the proposed scheme requires the receivers to store $2 \times \lceil \log_2 p \rceil$ bits and the sender to store $2 \times (k+1) \times \lceil \log_2 p \rceil$ bits. Each message has an authenticator of size $(k+1) \times \lceil \log_2 p \rceil$ bits. Authors proved also that k receivers cannot perform an impersonation or substitution attack with a probability greater than $1/p$ (p is chosen large enough so that it is hard for the attacker to make a good guess). S. Obana and K. Kurosawa [78, 94] derived lower bounds on the cheating probabilities (substitution and impersonation) and the sizes of keys of k out of n multi-receiver authentication schemes and showed that this scheme proposed by Desmedt et al. meets all their bounds with equality which means that this scheme is optimum. Nevertheless, this scheme is not practical for most of media-streaming applications (which are subject to time delivery constraints), since the sender has to generate and distribute polynomial shares for each message for each receiver. But this is the price to pay for unconditional security.

Safavi-Naini and Wang [120, 121] generalized Desmedt et al. polynomial scheme, in such a way that instead of a single message, each polynomial can be used to authenticate multiple messages. Then, authors proposed to construct k out of n authentication schemes using *cover-free set systems*. The main idea of the *cover-free set system* is that given a set of keys used by the sender to authenticate messages, how to affect subsets of these keys to receivers in a way that j $(j < k)$ fraudulent receivers cannot collaborate using their subsets of keys to cover the keys' subset of a group member. Similarly, Fujii et al. [55] proposed unconditionally secure broadcast authentication schemes based on *cover-free set systems*. Authors gave some combinatorial bounds and proposed some constructions that meet those bounds.

Conditionally secure authentication

In this approach an attacker is assumed to have finite resources and hence cannot make an attack that require complex computations that exceed its computation capacities.

CANETTI ET AL. PROTOCOL: The main idea behind the protocol proposed by Canetti et al. in [22] is that the sender appends to each multicast message M, l MACs using l different keys. Each receiver holds a subset of keys among the l sender's keys and verifies the authenticity of received messages using its subset of keys. For an adversary to forge a message of the valid sender, it needs to acquire the l keys from a coalition of w receivers. The cornerstone of the solution is that an appropriate choice of receivers' subsets of keys ensures that with high probability no coalition of up to w colluding bad members (where w is a parameter) know all the keys held by a good member, thus authenticity is maintained.

We denote by S the source of the transmissions and by u a receiver in the multicast group. The basic scheme proceeds as follows:

1. S maintains a set R of l keys, $R = \{K_1, \ldots, K_l\}$.

2. Each receiver knows a subset of this set of keys: receiver u knows the subset $R_u \subset R$.

3. When S sends a message M it first authenticates it with each of the l keys that it maintains, using a MAC. Then it sends the message M along with $MAC(K_1, M)|\ MAC(K_2, M)|\ \ldots|MAC(K_l, M)$.

4. Each receiver u verifies all the MACs which were created using the keys of its subset R_u. If any of these MACs is incorrect then u rejects the message.

The strength of this solution depends on the probability that the key subsets of a fraudulent coalition cover completely the key subset R_u of a given user u. In order to upper-bound this probability by a certain constant q, the authors suggested that the sender uses $l = 4e^{w \ln \frac{1}{q}}$ keys, where w is the size of the largest coalition, and that each key is chosen to a user's subset with probability $1/(w+1)$. According to this construction, authors proved that if the probability of computing the output of a MAC without knowing the key is at most q', then the probability that a coalition of w corrupt users can authenticate a message M to u is at most $q + q'$.

The proposed solution allows a relatively efficient generation and verification of the authentication information since it relies only on MAC's computations which are efficient to generate and verify. Authors propose to use MACs with a single bit as output. Hence, the authentication information is reduced to l bits. Boneh, Durfee and Franklin showed in [15] that the Canetti et al. construction has optimal length (up to a small constant factor) for an authenticator that is based purely on pseudo random functions. Each packet carries its authentication information and hence each packet is individually verifiable. Thus the solution tolerates packet loss. The main drawback of this solution is that it remains vulnerable to collusions of bad members.

6.1.2 Time Asymmetry

This approach consists in limiting the life time of keys used to authenticate multicast packets, in a way that if an attacker uses a key to compute a message authentication information on behalf of the valid sender, receivers reject its message because the key would have been expired. Therefore, the sender generates the keys periodically to authenticate multicast packets by interval of time. The following solutions deal with authenticating the keys themselves, and how to generate those keys, so that even if some keys are lost, the required keys are recovered from received keys to authenticate received packets. A common mechanism used by proposed solutions is *one-way key chains*. The following paragraph describes how such a key generation mechanism allows to authenticate the keys themselves and how to recover lost keys from received ones. Then we illustrate *Time Asymmetry* approach with some protocols.

ONE-WAY CHAINS: In order to use a MAC to authenticate a multicast message, the sender has to communicate a secret key to receivers in a secure way. Otherwise, any attacker can send a key to receivers and pretend to be the valid sender, then starts to send messages on the sender's behalf. Multicast streaming requires the *secure* transmission of keys to be *fast*. Therefore, it is not

affordable to sign each secret key. The main idea of *one-way chains* is to *certify* (using a digital signature for example) only a single secret. This *certified* secret is the culmination of recursive one-way computations that form a chain of secrets. Thus, the single certification allows to certify the whole one-way chain of secrets. Figure 6.3 shows the one-way chain construction. To generate a chain of length k, the sender picks randomly the last element of the chain K_k. Then it generates the chain by repeatedly applying the one-way function H. Finally, K_0 is the secret that should be sent securely (*certified)* to receivers. Then, the secrets are revealed in the inverse direction. To verify whether a received secret K_i is valid (comes really from the valid sender), the receiver of the secret checks that $H^i(K_i) = K_0$. Generally, given that K_j is valid, a received secret K_i is valid if $H^{i-j}(K_i) = K_j$. Moreover, if a secret K_i is lost, once a subsequent secret of the chain K_f $(f > i)$ is received, the lost secret K_i can be recovered by: $K_i = H^{f-i}(K_f)$.

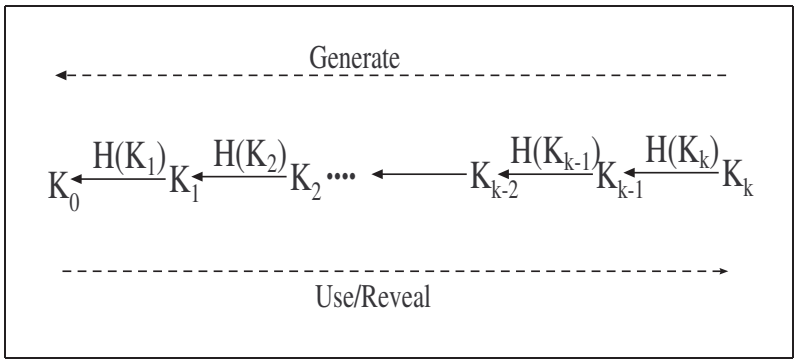


Figure 6.3: Generating MAC keys using a hash chain

Lamport [80] and Haller [60] used *one-way chains* in one-time password systems. In our case, the secrets of the chain correspond to MAC-keys and hence we use the expression *one-way key chains* to designate this key generation mechanism. Proposed solutions that use this mechanism, take advantage of the fast way that one-way chains allow to verify keys' authenticity as well as the fact that lost keys can be recovered once a valid subsequent key is received. Note that the "one-wayness" of the function H, used in the one-way chain construction, prevents receivers from generating valid future keys given a valid key.

TIMED EFFICIENT STREAM LOSS-TOLERANT AUTHENTICATION PROTOCOL (TESLA): Perrig et al. [104, 105] proposed the TESLA (Timed Efficient Stream Loss-tolerant Authentication) protocol. The main idea of TESLA is that the sender uses a different key in each interval of time to authenticate multicast messages sent within this interval of time. TESLA uses one-way key chains to generate the MAC keys. A secret MAC key used to authenticate multicast messages sent within a period of time is kept secret by the sender, to avoid that an attacker receives the key before other receivers and uses it to forge authenticated messages on behalf of the valid sender. When the period of time corresponding to the use of this key elapses, the sender discloses the key used to authenticate messages within this elapsed period. Then, receivers use this disclosed key to verify the authenticity of the previously received messages. If an attacker uses this key to forge a message on behalf of the sender, receivers will reject its message, because they have their clocks

synchronized with the sender's clock and they know then that the sender would have committed to another key corresponding to the following interval of time.

Figure 6.4 shows how TESLA generates and discloses MAC keys used in authenticating broadcast packets. At the top of the figure is the one-way key chain (using the one-way function H), and the

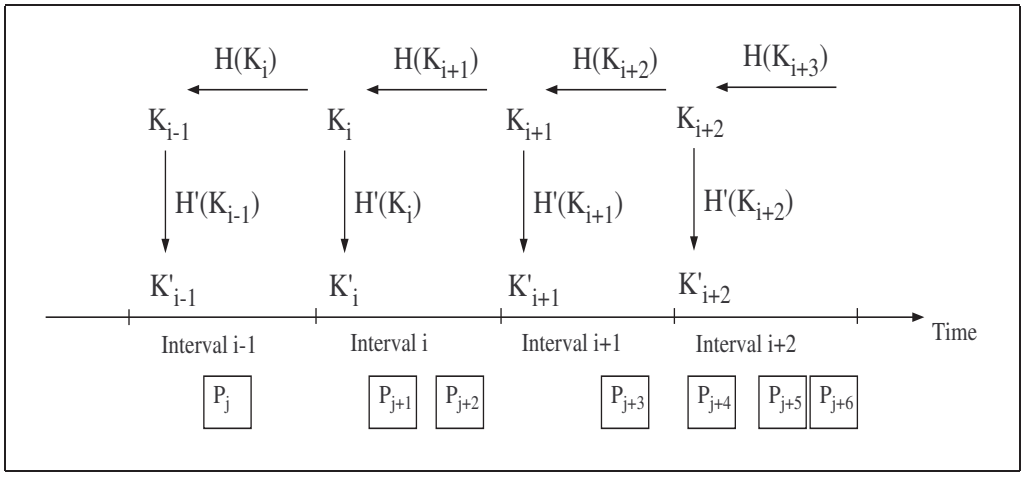


Figure 6.4: TESLA protocol

derived MAC keys (using the one-way function H'). Time advances left-to-right, and the time is split into time intervals of uniform duration. At the bottom of the figure, we can see the packets that the sender sends in each time interval. For each packet, the sender uses the key that corresponds to the time interval to compute the MAC of the packet. For example for packet P_{j+3}, the sender computes a MAC of the data using key K'_{i+1}. When a receiver receives packets corresponding to an interval of time, it buffers them and waits for the first packet of the following time interval (key disclosure). This packet will carry the key used to authenticate the buffered packets and hence allows their verification. In figure 6.4, assuming a key disclosure delay of two time intervals ($d = 2$), packet P_{j+3} would carry key K_{i-1} which will allow the verification of P_j.

With TESLA, the authentication information size is reduced to the size of one MAC. The *MACing* at the sender and the verification at receivers are very efficient. Packets can be individually authenticated and hence the protocol tolerates packets' loss. Besides, in case a key is lost, the chaining used in their construction allows to recover a lost key from subsequent keys. The main drawback of TESLA is its synchronization requirement between the source and receivers. This creates a new potential security hole for adversaries. Besides, if different users experience large differences in network propagation delays, many TESLA instances should be run to satisfy all those receivers in term of synchronization. Notice also that TESLA requires that received packets be buffered until their corresponding keys are disclosed and hence cannot be used with resource constrained devices or applications that require real-time transmission.

Perrig et al. proposed also some extensions in [103] to TESLA in order to cope with some of its drawbacks. Namely, authors proposed a modification that allows receivers to authenticate most packets as soon as they arrive by the mean of a slightly increase in the authentication information

size. They also proposed other modifications that improve the scalability of the scheme in the case where different receivers have different network propagation delays. Perrig et al. proposed in [107] a light version of TESLA called $\mu TESLA$ which is more suitable for ad hoc sensor networks that are known to be severely resource-constrained environments.

BERGADANO ET AL. PROTOCOL: Bergadano et al. [10, 11] proposed a solution using *time asymmetry*. The essence of the proposed solution is to authenticate each packet of data with a MAC using a key generated using a *one-way key chain*. The recursive relation between keys allows to recover lost keys and to verify the validity of received keys. The MACed packets as well as the keys used in calculating their MACs are multicast to receivers. To avoid that a fraudulent receiver uses a received key to forge a data packet on behalf of the legitimate sender, the sender guarantees that keys are known by receivers only when all receivers have received the packets authenticated with their respective keys. Therefore, receivers are synchronized with the sender's clock and instructed to reject any data packet whose MAC arrives late.

BINS AND BALLS PROTOCOL (BIBA): Perrig [102] proposed the *The BiBa one-time signature and broadcast authentication protocol*. The main idea of *BiBa* is that the sender uses a set of keys to authenticate messages. When the sender authenticates a message it discloses only a subset of keys that allow the verification of the packet's authentication information without being able to generate it. Hence, packets' authenticity is verified as soon as packets are received. However, if the same set of keys is used to authenticate a certain number of packets, each receiver would acquire the whole set of sender's keys after some subset disclosures, and hence would be able to forge messages on behalf of the valid sender. To avoid that, the sender changes the set of keys that it uses to authenticate messages periodically, before receivers would know all of them. The key generation uses *one-way key chain* mechanism, so that receivers can verify their authenticity and can recover them whenever some of them are lost.

Mitzenmacher and Perrig [88] proposed an improvement of the BiBa one-time signature scheme called *Powerballs*. Leonid Reyzin and Natan Reyzin [115] proposed also a one-time signature scheme based on one-way hash functions which is as faster as BiBa in verifying. Signing is even faster than verifying and the key and signature sizes are slightly improved.

6.1.3 Comparison

The efficiency of a data origin authentication protocol can be measured according to many criteria. Tables 6.1 and 6.2 compare some data origin authentication protocols (without non-repudiation) with respect to the following criteria:

Security Strength criteria

1. *Security Level*: corresponds to the fact that the authentication scheme relies on a *theoretic security model* (unconditional security), or on a *computational security model* (conditional security);

2. *Vulnerability to collusions*: corresponds to the fact that the authentication scheme fails under a collusion of fraudulent users.

QoS Criteria

1. *The latency at the source*: corresponds to the fact that the source needs to buffer packets before sending them.
2. *The latency at a receiver*: corresponds to the fact that a receiver needs to buffer packets before authenticating them.
3. *Tolerance to packet loss*: corresponds to the fact that the authentication process is possible even if some packets are lost.
4. *Authentication information size*: the size of the authentication information embedded to a packet.
5. *Synchronization required*: corresponds to the fact that receivers need to be synchronized with the source.

Protocol	**Security Level**	**Vulnerability to collusions**
Desmedt et al.	Unconditional Security	Yes
Canetti et al. Protocol	Conditional Security	Yes
Bergadano et al. Protocol	Conditional Security	No
TESLA	Conditional Security	No
BiBa	Conditional Security	No

Table 6.1: Comparison of some data origin authentication protocols with respect to *Security Strength criteria*

6.2 Multicast Data Origin Authentication with Non-repudiation

To assure data origin authentication and non-repudiation for a given message, the sender has to sign it using its private key. Hence, a naive solution to guarantee multicast data origin authentication with non-repudiation would be to sign each multicast message and then multicast the message as well as its signature. Upon receiving the message and its signature, receivers would be able to verify the message's data origin authenticity using the public key of the sender and non-repudiation is guaranteed. Existing digital signature mechanisms are very computationally expensive. Therefore, this naive solution raises the problem of the required computation power to sign and to verify each multicast message. Thus, this naive solution is not practical with most of media-streaming applications that require real-time transmission. Besides, power constrained devices cannot afford

Protocol	Latency at the source	Latency at receivers	Tolerance to packet loss	Authentication information size	Synchronization required
Desmedt et al. $(k,n)-MRA$	No	No	Yes	$(k+3)\times\lceil\log_2 p\rceil$ bits, where p is a large prime equal or greater than the number of messages	No
Canetti et al. Protocol	No	No	Yes	l bits, where l depends on the size of the largest fraudulent receivers coalition	No
Bergadano et al. Protocol	No	Yes	Yes	$\|MAC\|+\|k_m\|$	Yes
TESLA	No	Yes	Yes	$\|MAC\|[+\|k_m\|]$, where k_m is disclosed only once per period of time	Yes
BiBa	No	No	Yes	$k\times\|ball\|+\|counter\|[+\|k_m\|]$, where k is in the order of 15, $\|ball\|$ in the order of 64 bits	Yes

k_m: denotes a MAC key. $|x|$: denotes the size of x.

Table 6.2: Comparison of some data origin authentication protocols with respect to some QoS criteria

the required energy to guarantee the operation of this naive solution. In what follows, we present three alternatives that aim to cope with limitations of this naive solution.

6.2.1 Signature propagation

The essence of the schemes in this category is to make each packet carrying authentication information of other packets and this process culminates into a small piece of information which is signed. Hence the single signature effects propagate throughout the packets' relationship. This propagation assures data origin authentication and non-repudiation of the whole related packets. Figure 6.5 further classifies proposed solutions regarding tolerance to packet loss.

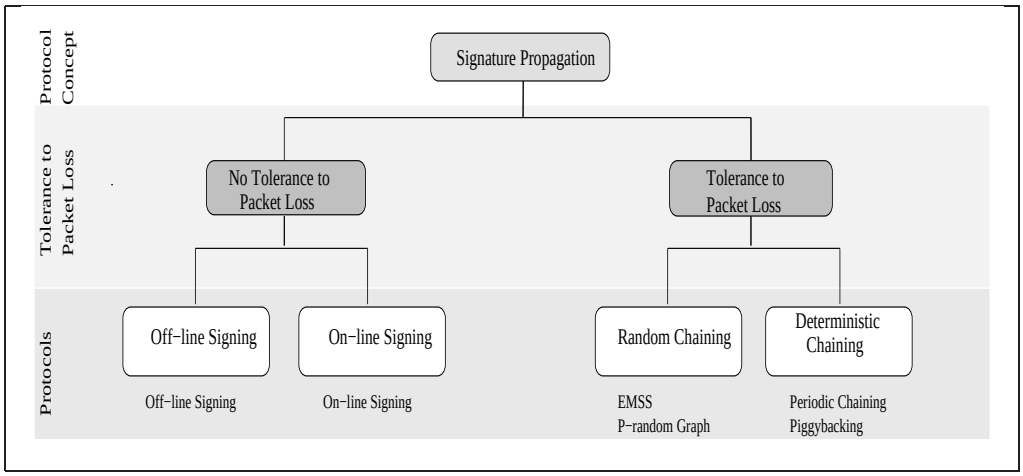


Figure 6.5: Taxonomy refinement for *Multicast Non-repudiation* using *Signature Propagation Concept*

Schemes not tolerant to packet loss

Some data-streaming applications use a reliable transport layer such as TCP and hence packet loss is not a concern for data origin authentication in this kind of applications. An example of applications that can be modeled as streaming is *executing applets* [57]. Applets are organized into modules which are downloaded and executed module by module. In this case instant data origin authentication is required to avoid an attacker sending malicious code, and non-repudiation is also required in order to proof at the justice the identity of the responsible for sending malicious code.

SIMPLE OFF-LINE CHAINING: The main idea of the solution proposed by Gennaro and Rohatgi in [56, 57] is to divide the stream into blocks and embed some authentication information in the stream itself. The authentication information of the i^{th} block is used to authenticate the $(i+1)^{st}$ block. This way the signer needs to sign only the first block and then the properties of this single signature will *propagate* to the rest of the stream through the authentication information.

In this solution, it is assumed that the sender knows the entire stream in advance (off-line). To authenticate the stream, the sender embeds in the current block a hash of the following block which in turn includes the hash of the following one and so on.... Then the sender signs only the first block. To verify the stream's data origin authenticity, a receiver first receives the signature on the hash of the first block. After verification of the signature, the receiver knows what the hash of the first block should be and then starts receiving the full stream and starts verifying stream's authenticity block by block by comparing the hash of the current block with the hash received in the previous block. Figure 6.6 illustrates the construction of the authentication information chain for a stream composed of n blocks $\{b_n, b_{n-1}, \ldots, b_2, b_1\}$. The sender constructs, off-line, the authentication chain $\{B_n, B_{n-1}, \ldots, B_2, B_1\}$ such as $B_i = b_i | H(B_{i-1})$. The first block of the authentication chain B_1 is signed using the private key of the sender SK.

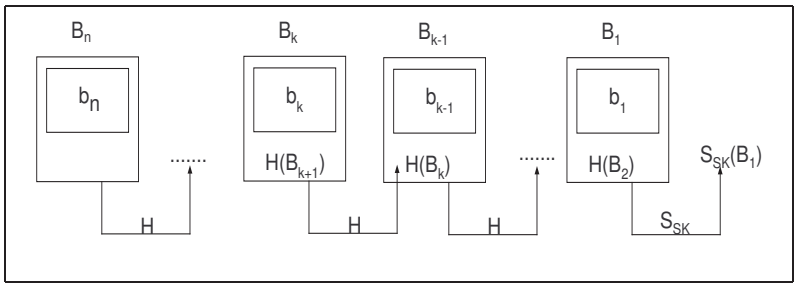


Figure 6.6: Simple off-line chaining (Example)

A receiver verifies first the signature of the first block: $S_{SK}(B_1)$. Then, the receiver authenticates the block B_i using the hash $H(B_i)$ received in B_{i-1}. And hence, all the blocks of the stream are recurrently authenticated.

The major merit of this proposition, is that authors introduced and *proved* the security (data origin authentication and non-repudiation) of the hash-chaining scheme. With this solution, the authen-

tication information is reduced to one hash per block. The sender signs only the hash of the first block. The other blocks are authenticated using only a hash computation. Receivers can verify block's authentication as soon as the block is received. However, this solution is not fault tolerant: if a block is lost, the authentication chain is broken and hence all subsequent blocks can no longer be authenticated. Besides, the fact that all the stream should be known to be able to construct the chain (*get-all-before*) is a too strong requirement for practical Internet applications.

The authors proposed also *On-line chaining* in [56, 57]. The essence of this second solution is to make each data block carrying the 1-time public key used to 1-time sign the following block. Hence, by signing the first 1-time public key, the signature is amortized over the stream. The major contribution of this solution is to get rid of the *get-all-before* requirement of the previous solution.

Schemes tolerant to packet loss

Most of media-streaming applications do not use a reliable transport layer since they require real-time transmission and hence packets' re-transmission cannot be used to recover lost data. The general concept used to cope with packet loss in this category of data origin authentication schemes is to create redundancy in the packets' authentication information so that even if some packets are lost, the required authentication information is recovered from received packets. In other words, instead of embedding the hash of a packet only in the next (or previous) packet, the packet's hash is embedded within several packets. These *embeddings* of packet's hash into other packets to create authentication information redundancy form a *topology of packet relations*. This technique is subject to the following performance law: *the more important is redundancy, the higher is robustness against packet loss*. But we know also that *the more important is redundancy, the higher is bandwidth consumption*. Therefore, the aim of proposed solutions using authentication information redundancy approach is to define the best *topology* that minimizes redundancy on one hand and resists better to packet loss on the other hand. We further distinguish between protocols that use *random hash-chaining*: such as Efficient Multi-chained Stream Signature (EMSS) proposed by Perrig et al. in [104] and p-random graphs proposed by Minner and Staddon in [86], from those using *deterministic hash-chaining*: such as Periodic chaining proposed by Modadugu and Golle in [58] and Piggybacking proposed by Miner and Staddon in [86] (cf. taxonomy refinement in figure 6.5).

EFFICIENT MULTI-CHAINED STREAM SIGNATURE (EMSS): Perrig et al. [104] introduced the notion of *redundant hash-chaining* which means that each packet of the stream is *hash-linked* to several *target* packets. Thus, even if some packets are lost, a received packet is verifiable if it remains a hash-link path that relates the packet to a signature packet. For a given packet, EMSS chooses target packets randomly. Hence, EMSS provides more or less probabilistic guarantees that it remains a hash-link path between the packet and a signature packet, given a certain rate of packet loss in the network.

EMSS operates as follows: when a packet is presented to be sent, the sender embeds some hashes of other packets in this packet and computes the overall hash code. This hash code is buffered to

be included later in d target packets chosen randomly by the sender (where d is the *redundancy degree*). Figure 6.7 shows an example where the redundancy degree d is equal to two. In this example, packets P_i, P_{i+1} and P_{i+2} are verifiable because they have a hash-link path to the signature packet S_j.

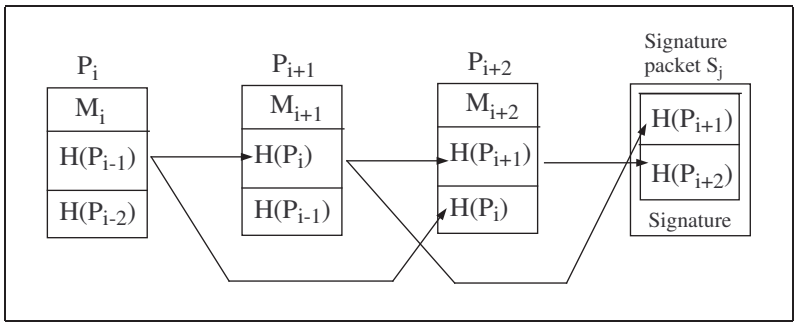


Figure 6.7: EMSS hash chaining with redundancy degree equal to 3

In order for the sender to continuously assure the authentication of the stream, the sender sends periodic signature packets. To verify authenticity of received packets, a receiver buffers received packets and waits for their corresponding signature packet. The signature packet carries the hashes that allow the verification of few packets. These latter packets carry, in turn, the hashes that allow to verify other packets, and so on until the authenticity of all received packets is verified.

EMSS provides probabilistic robustness against packet loss. Authors showed using simulations that using a redundancy degree equal to six, more than 90% of received packets can be verified (hold a hash-link path to a signature packet) even if 60% of stream packets are lost. The main drawback of this scheme is that receivers experience latencies before varifying received packets, because of waiting for the signature packet corresponding to the received packets. Besides, the periodic signing makes the solution not suitable for most of energy constrained devices.

P-RANDOM GRAPHS: Minner and Staddon [86] proposed a redundant and random hash-chaining scheme to tolerate packet loss in a network where each packet is lost independently at random with probability q. Authors were interested in applications in which the sender has a priori knowledge of the content. Therefore, the *hash-link topology* is constructed before the first packet of the stream is sent. The *random redundant topology* proposed by the authors is called *p-random graph*. In a basic *p-random graph* scheme, packets of the stream are numbered from 1 to n. P_1 is the *signature packet*, and for all pairs of packets (P_i, P_j) where $j < i$, the hash of packet P_i is embeded within packet P_j with probability p. Once the p-random graph of the stream is constructed, the packets of the stream are sent respectively. A receiver starts by receiving the *signature packet*. If it is valid, the receiver verifies subsequent packets *on the fly* by checking the existence of a *hash-link path* between the received packet and the *signature packet*.

PERIODIC CHAINING: Modadugu and Golle [58] proposed to use a similar strategy to EMSS, but packets that will carry the hash of a given packet are chosen in a *deterministic* way rather than randomly. The proposed *deterministic topologies* of packet hash-chains (called *Augmented*

chains) are designed to be optimized to resist a *burst loss*. The goal of the proposed schemes is to maximize the size of the longest single burst of loss that the authentication scheme can withstand (Once few packets have been received after a burst, the scheme recovers and is ready to maintain authentication even if further loss occurs).

PIGGYBACKING: Miner and Staddon [86] proposed a similar authentication scheme (called *Piggybacking*), based on hash chaining techniques, specifically designed to resist multiple bursts. The proposed scheme deals with the case where data carried by different packets has more or less importance from the point of view of the application level. Thus, packets are organized into classes with different priorities. Then hash chaining is made in a way that: the higher is the priority of a class, the more redundant is hash-chaining of packets belonging to that class, in order to resist more against bursty losses.

6.2.2 Signature Dispersal

The *authentication information dispersal* schemes base on *processing* authentication information and dispersing it throughout a set of packets. This processing is made in a way that even if some amount (that does not exceed a certain threshold) of this processed information is lost, the whole authentication information can be recovered from received data. Let us consider the following intuitive example to clarify the aim of this approach. We illustrate *Signature Dispersal* approach with the following protocols:

STAR/TREE CHAINING: Wong and Lam [139, 138] proposed a set of schemes where authentication information is replicated through the stream packets in a way that each packet carries the whole authentication information required to be individually verifiable. The following example illustrates the basic star chaining scheme: let us consider a sequence of n packets to be signed. Instead of signing each packet, the sender computes a single signature per block of packets as follows: first, the sender computes the hash of each packet. Then it computes the digital signature over the concatenation of these latter hashes. We call this signature: the *block signature*. Figure 6.8(a) illustrates this authentication information generation phase for a block of three packets.

Then, to each packet is appended the hashes of the other block packets in addition to the block signature (cf. figure 6.8, step 3). Therefore, even if all packets but one are lost, the received one can be verified since it carries all the required information to allow its verification. Indeed, to verify a received packet, its hash is computed. Then this hash is concatenated to the other hashes carried by the packet following the original sequence order (fig. 6.8, step 4). This concatenation of digests is then verified against the signature, carried also by the packet, using the public key of the sender (fig. 6.8, step 5).

We notice in this simple example, that *authentication information processing* is reduced to *replication*. In fact, each packet carries almost a whole copy of the authentication information consisting in a signature with a concatenation of packets' hashes. We notice also that the tolerated *packet loss threshold* is $n - 1$ lost packets out of n sent packets. This is the extreme case, where each

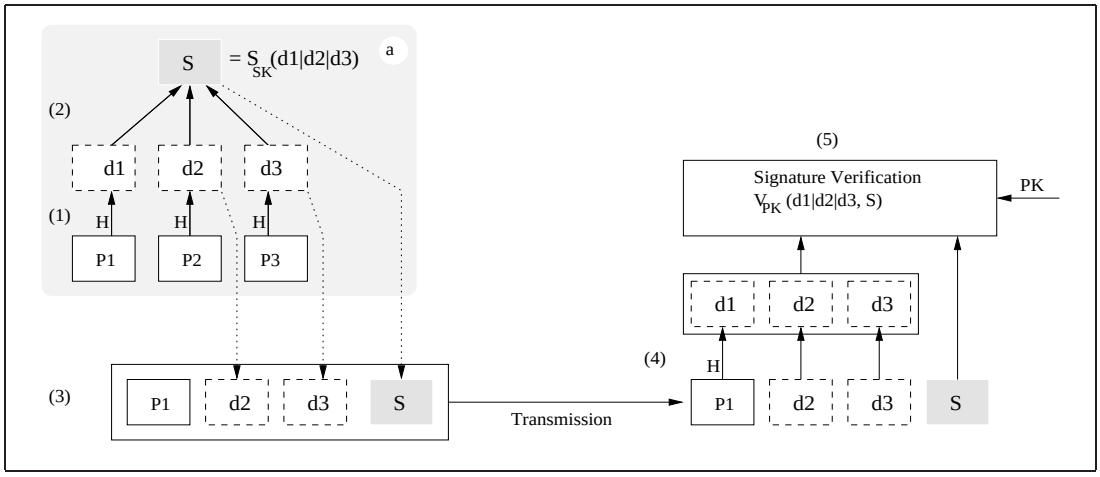


Figure 6.8: Signature Dispersal (Example)

packet is *individually verifiable*. But the price to pay is the large size of authentication information share carried by each packet. Since in practice, we know that only a portion of packets is lost (let say, k out of n packets are lost with $k < n$), the authors proposed *Tree Chaining* scheme where the digests of block packets are organized into a tree that allows to reduce the amount of the carried authentication information by each packet to $O(\log_2(n)$ instead of $O(n)$ in the *Star Chaining* scheme, where n is the number of packets in a block.

SIGNATURE AMORTIZATION USING IDA (SAIDA): In order to reduce the size of the authentication information carried by each packet, the approach proposed by Park et al. [98, 99] uses a famous mechanism, called IDA [1], to disperse the n hashes of n block packets as well as the block signature into n pieces in such a way that the n pieces (and hence the n hashes as well as the block signature) can be reconstructed even if $n-m$ pieces are lost (m being a parameter). The proposed protocol proceeds as follows:

1. The stream is divided into blocks of n packets, denoted by $B_1, B_2, \ldots, B_i, \ldots$;
2. The sender computes the packet hashes in a block B_i using some hash function H. Then it concatenates them to form $F_i = H(P_1^i)|H(P_2^i)|\ldots|H(P_n^i)$, where P_j^i is the j^{th} packet in the block B_i.
3. Then, the sender signs the block B_i by signing F_i (the hashes' concatenation). We denote this block signature by: S_i.
4. The source applies then the IDA algorithm to both the hashes F_i and the signature S_i, and thus we get the IDA-pieces $(F_i^j)_{1\leq j\leq n}$ and $(S_i^j)_{1\leq j\leq n}$ (respectively), where F_i^j is the j^{th} IDA-piece of F_i, and S_i^j is the j^{th} IDA-piece of S_i.

[1] IDA has been proposed by Rabin in [112]. It was originally developed to provide safe and reliable storage or transmission of information in distributed systems. The basic idea of IDA is to process the original data denoted by F, by introducing some amount of redundancy and splitting the result into n pieces, which are then transmitted. Reconstruction of F is possible with any combination of m pieces (assuming $n-m$ pieces are lost during transmission), where $m \leq n$.

5. To each packet P_j in the block B_i, the sender appends the corresponding IDA-pieces F_i^j and S_i^j to form a block of authenticated packets (cf. figure 6.9).

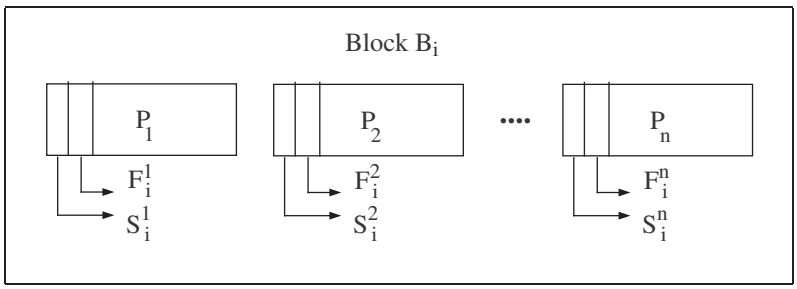


Figure 6.9: Authenticated packets using IDA

6. The sender can now send the authenticated packets, and the scheme tolerates that $n - m$ authenticated packets be lost.

7. Now, by applying the IDA-algorithm, a receiver can reconstruct the authentication information (packets' hashes and the block signature) to verify packets using only at least m received authenticated packets.

Authors showed that with SAIDA protocol, the authentication information size is reduced to $\frac{n}{m} \times |F_i|$, where F_i is the concatenation of the n packets' hashes of block B_i. This protocol allows to achieve a trade-off between the authentication information overhead (bandwidth) and tolerance to packet loss (parameter m). However, this scheme requires a high computation overhead due to the IDA processing. Besides, it introduces delays at both the sender and receivers to generate and verify the authentication information using IDA.

Pannetrat and Molva [96, 97] proposed a stream authentication scheme that is similar to the above one [98]. They propose to authenticate real-time data streams by piggybacking the current block's encoded authentication information (via an erasure code) onto: the next block, the previous block or itself, depending on the available memory at the sender or receivers side. Lysyanskaya et al. proposed also in [81] a protocol using the same technique.

6.2.3 Differed signing

Instead of signing data itself, with *Differed Signing* the sender signs a small piece of information consisting of one-time keys in a way that this *signing does not interfere with real-time transmission* required by most of media-streaming applications. These signed one-time keys are then used to one-time sign data itself *in a fast way*. Notice that the first conventional signing step allows to certify the authenticity of the one-time keys used in signing data. The following protocol illustrates this approach.

ON-LINE / OFF-LINE DIGITAL SIGNATURES: Shimon Even et al. [51, 52] suggest that slower pre-computations using conventional digital signature mechanisms can be tolerated, provided that

they do not have to be performed on-line (i.e., once the message to be signed is handed to the signer and while the verifier is waiting for the signature). The authors introduced the notion of an on-line / off-line signature scheme, in which the signing process can be broken into two phases:

1. The first phase, performed off-line, is independent of the particular message to be signed. This phase consists of generating a pair of one-time signing / verifying keys: (sk, pk), and producing an ordinary signature of the one-time verification key $S_{SK}(pk)$ (where SK is the sender's conventional private key). Both, one-time keys and the signature are stored for future use in the on-line phase.

2. The second phase is performed on-line, once the message M is presented. It consists of retrieving a precomputed unused pair of one-time keys (sk, pk), and using the one-time signing key to sign the message: $\sigma_{sk}(M)$. The corresponding one-time verification key and its precomputed signature are attached to the one-time message signature to produce the final signature: $pk|S_{SK}(pk)|\sigma_{sk}(M)$.

To verify that the triple $pk|S_{SK}(pk)|\sigma_{sk}(M)$ is indeed a signature of M with respect to the public key PK of the sender, the receiver acts as follows:

1. First, the receiver checks that $S_{SK}(pk)$ is indeed a signature of the one-time verification key pk with respect to the public key PK.

2. Next, the receiver checks that $\sigma_{sk}(M)$ is indeed a one-time signature of the message M with respect to the one-time verification key pk.

Since each message is one-time signed independently from other messages, the proposed scheme tolerates packet loss, and received messages can be verified as soon as they arrive to a receiver. The main drawback of the proposed scheme is that a receiver has to verify a conventional digital signature on a one-time verification key in addition to a one-time signature on the received message. The conventional digital signature verification may be not suitable for most of resource constrained devices. Besides, it is well known that one-time signatures size may be very large, and hence the proposed scheme may lead to a high bandwidth consumption. The off-line phase can raise some limitations when the message arrival rate is very high, especially with streaming-media applications such as live video-broadcasting. The best solution to this problem, is to parallelize the solution in a way that the off-line expensive phase be processed by a powerful server.

Pankaj Rohatgi [119] proposed to have the off-line computation create and sign *k-time* key pairs instead of one-time key-pairs, so that the cost of the most expensive operation, i.e. signing k-time verification keys using conventional digital signing, can be amortized over k-signatures. With this scheme, the multicast sender has an off-line process which generates k-time key-pairs and signs the k-time verification keys using the sender's long term conventional signature key. The sender uses each such k-time key-pair to sign k successive messages in the on-line phase. Since the main drawback of one-time / k-time signatures is the large size of produced signatures, the authors

proposed some techniques to reduce the size of k-time signatures but then speed of the underlying mechanisms is also decreased.

6.2.4 Comparison

The efficiency of a data origin authentication protocol with non-repudiation can be measured according to many criteria. Table 6.3 gives a comparison of some data origin authentication with non-repudiation protocols with respect to the following criteria [2]:

1. *The latency at the sender*: corresponds to the fact that the sender needs to buffer packets before sending them.

2. *The latency at a receiver*: corresponds to the fact that a receiver needs to buffer packets before authenticating them.

3. *Tolerance to packet loss*: corresponds to the fact that the authentication process is possible even if some packets are lost.

4. *Authentication information size*: the size of the authentication information embedded to a packet.

Protocol	Latency at the source	Latency at receivers	Tolerance to packet loss	Authentication information size
Off-line chaining	Yes	No	No	$\|d\|$
On-line chaining	No	No	No	$\|pk\| + \|\sigma\|$
EMSS	No	Yes	The authentication probability of a packet is at least 90%	$6\|d\|$
p-random graphs	Yes	No	Pr(P_i is verifiable $\mid$ P_i is received)$\geq 1-(1-p)(1-(p(1-q))^2)^{i-2}$	$p(n-1)$ hashes per packet in average
Periodic chaining C_a	No	Yes	Yes: Each block of packets tolerates a single burst of length up to $(a-1)$	$2\|d\|$
Augmented Chain $C_{a,p}$	Yes	Yes	Yes: Each block of packets tolerates a single burst of length up to $p(a-1)$	$2\|d\|$ in average
Piggybacking	Yes	Yes	Yes: Each prioritized packets' set S_i tolerates x_i bursts of b_i packets	$\|d\| \times (x_i+1)$ at least, for a packet in class S_i
Tree chaining	Yes	No	Yes	$(\log_k(n)-1)\|d\|+\|S\|$, where k is the degree of the constructed hash tree
SAIDA (n,m)	Yes	Yes	Tolerates up to $n-m$ lost packets	2 IDA-pieces = $\frac{\|S\|+n\times\|d\|}{m}$
On-line/Off-line Signing	No	No	Yes	$\|\sigma\| + \|S\| + \|pk\|$

$|d|$: size of a digest (hash). n: number of packets in a block. $|S|$: size of a signature. $|\sigma|$: size of a 1-time signature. $|pk|$: size of a 1-time public key. q: loss probability of a packet.

Table 6.3: Comparison of some *data origin authentication with non repudiation* protocols

[2] With EMSS, we consider results of the special case simulated by authors

6.3 Conclusions

In this chapter, we gave an overview of the problems relating to data origin authentication in group communication. After having presented a classification of existing protocols, we described many protocols within each identified class. We have also dressed a comparison between typical solutions regarding a set of important criteria. This survey allowed us to come up with many interesting conclusions: first of all, data origin authentication is a required component in the whole multicast security architecture. But, many challenges obstruct the design of a data origin authentication scheme: the large number of multicast group members and the important data volume conveyed by multicast applications require scalability. Since many applications use unreliable transport layer, the authentication scheme should tolerate packet loss (the authentication process should be possible even if some packets are lost). Moreover, receivers may have limited resources in terms of computation and storage (such as PDAs and notebooks) and hence the data origin authentication scheme should not assume high availability of such resources at receivers. It is difficult to satisfy the overall constraints and requirements involved in data origin authentication and hence there is no a best solution but there are good solutions regarding specific requirements and features. Multicast data origin authentication has matured over the last twenty years, but there remain open problems in the area that must be resolved to allow a larger deployment of multicast applications that require data origin authentication.

Efficient Multicast Non-repudiation in Real-time Transmission: Most of proposed solutions focused on trading tolerance to packet loss for bandwidth overhead. All of these solutions require latencies at either the sender or receivers. Development of non-repudiation mechanisms without the requirement of packet buffering remains an open issue.

Efficient Multicast Non-repudiation in many-to-many Communications: The problem above becomes worse when considering many-to-many communications. With current proposed schemes, receivers would have to manage packet buffering for each source. Besides storing senders' public keys may be an issue for resource constrained devices.

Efficient Multicast Non-repudiation in Severely Resource Constrained Networks: Although there have been some attempts to deal with data origin authentication in severely resource constrained networks (such as sensor networks), non-repudiation in this type of networks remains a challenging problem because of the expensive underlying cryptographic mechanisms.

Mobility-aware Multicast Data Origin Authentication: When considering mobile multicast receivers, the collusion problem becomes more relevant. Besides, time asymmetry approaches are not efficient in such settings, because of packet propagation delay variations due to *topology changes* in such networks.

Adaptability to packet loss: Packet loss rate may change over time depending on the incidents and congestions in the network. Furthermore, the repartition of packet loss throughout a large network is not uniform.

In the following chapters, we propose protocols that take into consideration the variation of packet loss rate in order to better trade tolerance to packet loss for other performance requirements.

Chapter 7

Source driven Adaptive Hash-chaining

MANY protocols have been proposed to assure data origin authentication of a multicast flow with non-repudiation of the origin relying on *signature amortization* scheme, which uses *hash-chaining* techniques. The signature and its amortization induce some *extra-information* called the *authentication information*. Besides, most of multicast media streaming applications do not use reliable transport layer. Hence, some packets may be lost during their transmission. Therefore, the proposed solutions introduce *redundancy* in the *authentication information*, in such a way that even if some packets are lost, the required authentication information can be recovered in order to verify received packets' authenticity. In this case, the *bandwidth overhead*, induced by the redundant authentication information, increases. The existing solutions deal with how to trade bandwidth for tolerance to packet loss.

In this chapter, we propose a new adaptive and efficient protocol called: *Hybrid Hash-chaining scheme for Adaptive multicast data origin authentication* (H_2A) which authenticates the source of a multicast flow, assures non-repudiation and tolerates packet loss. In contrast to other protocols [57, 58, 104, 86] based on static hash-chaining, with our protocol we propose a new *hybrid and adaptive hash-chaining* technique which adapts the *redundancy chaining degree* (the amount of authentication information) depending on the actual *packet loss ratio* in the network. Besides, this new hash-chaining technique combines deterministic hash-chaining with random hash-chaining, in contrast to existing protocols that use either deterministic [58, 86] or random hash-chaining [104]. We carried out simulations using NS-2, and show that the adaptation of the *redundancy degree* allows to save bandwidth, and the combination of the random with deterministic hash-chaining allows to increase the robustness to packet loss. This chapter is based on [26, 27, 29].

In what follows, we present our protocol which uses the concept of amortizing a single digital signature over multiple packets using hash-chaining in such a way that reduces the bandwidth overhead and enhances the verification ratio of received packets even if some packets are lost. Finally, we evaluate and compare it with other protocols using NS-2 simulations.

7.1 H_2A: Hybrid Hash-chaining scheme for Adaptive multicast data origin authentication

To achieve non-repudiation, we rely on a conventional signature scheme for example RSA [117]. Unfortunately, the computation and communication overhead of current signature schemes is too high to sign every packet individually. To reduce the overhead, one signature needs to be amortized over multiple packets. The amortization is achieved using *hash-chaining*, which consists in signing a single packet and amortizing this single signature over multiple packets by *hash-linking* the current packet to another packet in the stream. In section 6.2.1 we discussed a basic chaining scheme. In our protocol, we use a *redundant hash-chaining* scheme to tolerate packet loss. Moreover, the *redundancy degree* of our *redundant hash-chaining* scheme is *adaptive* with respect to the *actual* packet loss ratio in the network. This *adaptation* of the *redundancy degree* allows to save *bandwidth overhead* compared to *static redundancy degree*. In the following paragraphs, we present some terminology then we detail the hash chaining technique used in our protocol. Then, we describe the operation of the (H_2A) protocol.

7.1.1 Terminology

We define some terminology to simplify the following discussion: if a packet P_j contains the hash of a packet P_i, we say that a **hash-link** connects P_i to P_j, and we call P_j a **target** packet of P_i. A **signature packet** is a sequence of packet hashes which are signed using a conventional digital signature scheme. A hash-link relates a packet P_k to a signature packet S_l, if S_l contains the hash of P_k. We designate by **redundancy degree** the number of times that a packet hash is embedded in subsequent packets to create redundancy in chaining the packet to a signature packet. A packet P_i is **verifiable**, if it remains a **path** (following the hash-links) from P_i to a signature packet S_j (even if some packets are lost). We designate by **verification ratio**: the number of verifiable packets by the number of received packets. The verification ratio is a good indicator of the **verification probability** which means the probability for a packet to be verifiable given that it is received: $P(\text{packet is verifiable}/\text{packet is received})$. This probability is equal to the probability that it remains a **hash-link path** (a hash-chain) that relates the packet to a signature packet.

7.1.2 Redundant and Hybrid Hash-chaining scheme

The basic idea of hash-chaining is that each packet carries the hash code of the previous packet. A final packet (the signature packet) is signed and guarantees data origin authentication and non-repudiation of the chained packets [57]. In order to tolerate packet loss, we make *redundant hash-chaining*: instead of carrying a single hash of the previous packet, each packet carries the hashes of multiple packets, so that even if some packets are lost, there is a *probability* that it remains *hash-link* paths between received packets and the signature packet. If a *hash-link* path exists between a

received packet and the signature packet, then the authenticity of the received packet is *verifiable* [57, 104].

When a packet is presented to be sent at the sender, it is *hash-linked* to k subsequent packets following the two steps:

(a) *Deterministic Target Packet:* in this step, the hash of the current packet is embedded into the next packet *systematically*. What motivates this choice of the target packet is the *bursty* nature of packet loss [100]. Indeed, it is easy to see that since packets are lost in a bursty way, the received packets are also received contiguously. Hence, if each packet is chained systematically to its subsequent packet, then if only one packet is verifiable then all the packets that follow it (in the same contiguous received segment) are also verifiable. In figure 7.1, packets P_{f-1} to P_{f-n} are verifiable because P_f is verifiable (it holds a path to the signature packet).

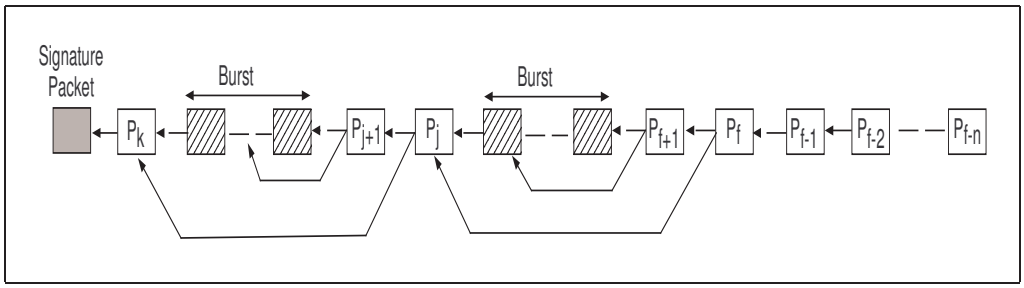


Figure 7.1: Hybrid hash-chaining impact on verification probability

(b) *Random Target Packets:* in this step, the hash of the current packet is embedded within $k-1$ subsequent packets chosen *randomly*. What motivates this *random hash-chaining* is the results of Perrig et al. in [104] that show that random hash-chaining allows to reach high verification ratio in bursty packet loss model.

This combination of deterministic with random hash-chaining achieves better verification ratio compared with purely random hash-chaining. Indeed, we carried out simulations to compare the resistance to packet loss of *purely random hash-chaining* with *hybrid hash-chaining*. We used a two redundancy degree per each technique, and we varied packet loss from 10% to 40%, then we noted the *verification ratio* achived by each technique. As we can see in figure 7.2, *hybrid hash-chaining* resists better to packet loss: when the packet loss ratio reaches 40%, the hybrid scheme maintains a verification ratio greater than 90% while the purely random scheme drops to 60%.

7.1.3 Adaptive Redundancy Degree

In contrast to existing protocols [104, 58, 86], the *redundancy degree* k in our protocol is *adaptive* and depends on the *actual* packet loss ratio in the network. Indeed, using simulations, that we

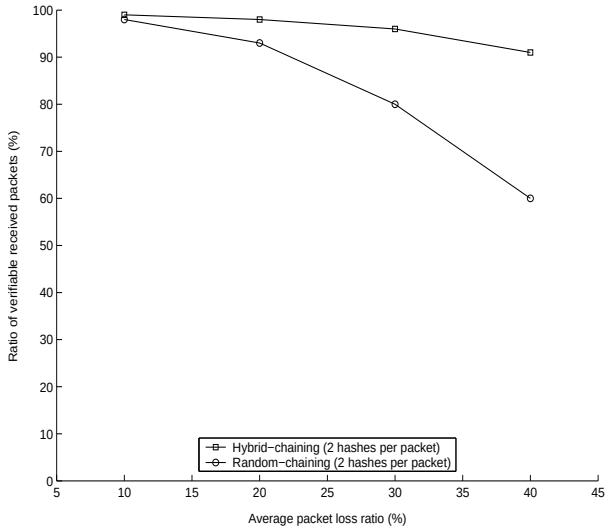


Figure 7.2: Robustness against packet loss: hybrid vs. only random hash-chaining

(k)	The redundancy degree
(f)	Number of packets after which a signature packet is sent
(d)	The scope within which packets are chosen randomly to include the hash of a packet
(b)	The average length of bursts in a bursty packet loss pattern
(t)	The period of time after which the quality of reception reports are sent
(θ)	The period of time after which the source analyzes the received quality of reception reports to update the redundancy degree (k)
(v)	The desired verification ratio of received packets

Table 7.1: H_2A parameters

detail in subsequent sections, we noticed that required redundancy degree to reach 99% of verification ratio depends *proportionally* on the packet loss ratio (cf. figure 7.9). Therefore, we suggest to exploit receivers' feedback regarding packet loss in the network to adapt the redundancy degree and hence to use only the required amount of authentication information to reach the best verification ratio. We assume that there exists a means for receivers to communicate to the sender the packet loss ratio in the network (for example by sending periodic RTCP [123] Receiver Reports). Relying on this receivers' feedback, the source decides what is the best redundancy degree to use in order to tolerate the actual packet loss ratio in the network.

7.1.4 H_2A protocol

The different messages exchanged between a source of an authentic data-stream and a receiver are shown in figure 7.3. It illustrates also the periodic operations executed by the source and the receiver to adapt the redundancy degree and to verify the authenticity of received packets respectively. Further explanations and algorithms are given in the following paragraphs. In table 7.1, we summarize the parameters involved in H_2A protocol. These parameters influence the

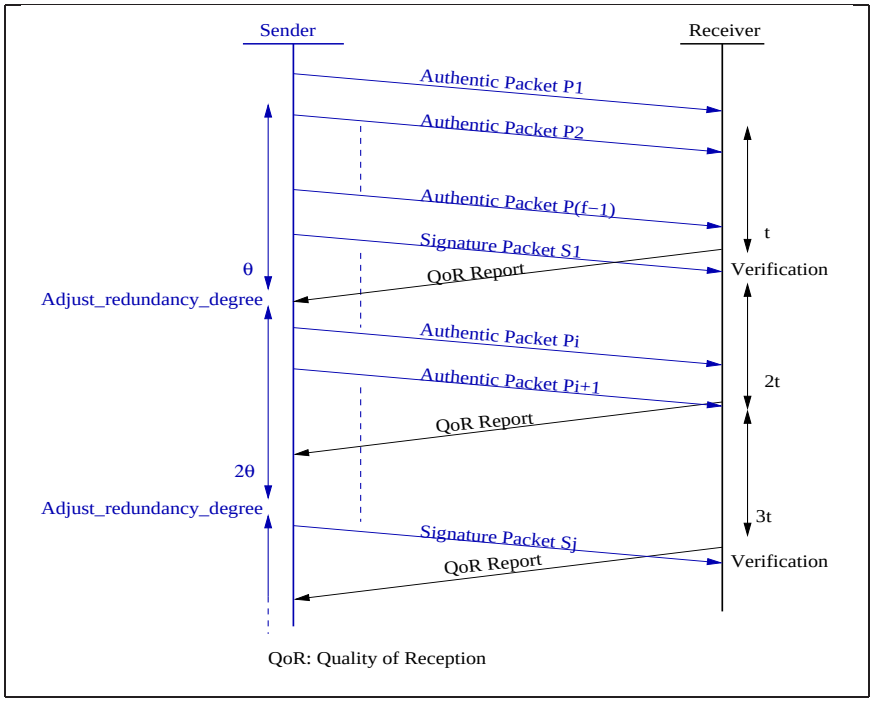


Figure 7.3: H_2A Sequence Diagram

computation and communication overheads, the delay until verification, and the robustness against packet loss. We want to achieve low overhead while maintaining high robustness against packet loss. In our case, we fixed these parameters by the means of empiric tests through simulations. In real settings, it would be the task of administrators to fix them depending on the available host and network resources. For instance, the increase of the parameter f induces more storage overhead at receivers, in order to buffer received packets before verification. Similarly, the decrease of the periods t and θ, increases the bandwidth overhead induced by the transmission of quality of reception reports, while it would bring more information to the sender to take instant decisions regarding the most suitable redundancy degree.

The Sender Side

A source of a stream applies the hash chaining technique described above for each packet P_i before it sends it. Figure 7.4 shows an example, where the redundancy degree is equal to 3. After each f data packets, the source sends a signature packet. These periodic signature packets allow to assure continuous non-repudiation of the stream. Besides, since the verification process depends on the reception of signature packets, the source replicates signature packets so that their loss probability becomes very low. After each period of time θ seconds, the source analyzes the received quality of reception reports and adjusts the redundancy degree k accordingly to maintain the desired verification ratio v. The algorithm at the source would then be as shown in figure 7.5. The *adjust_redundancy_degree* function determines the best redundancy degree k to reach the desired verification ratio v given that packets may be lost in the network with an average ratio

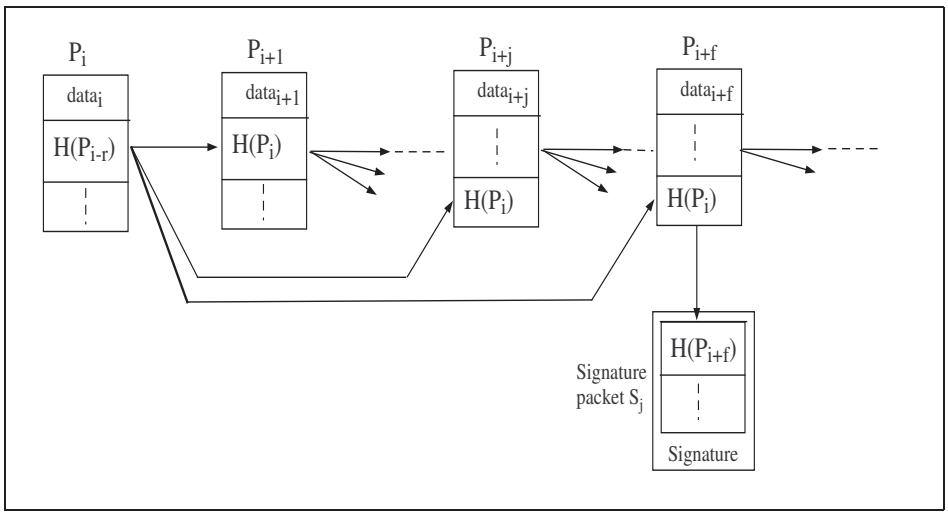


Figure 7.4: H_2A hash chaining example

```
SENDER ALGORITHM
for each packet P_i do
   include H(P_i) = h_i in the packet P_{i+1};
   do k − 1 times
      generate a random number j so that j ∈ [i + 2, i + d];
      include H(P_i) = h_i in the packet P_{i+j};
   end.
   send P_i;
end.

after each f packets do
   sign the current packet S_l;
   send the signature packet S_l;
end.

upon timeout do
   compute the average packet loss ratio
     from the received quality of reception reports:  avg;
   k=adjust_redundancy_degree(avg, v);
   schedule timeout after θ seconds;
end.
```

Figure 7.5: The algorithm at the source side

equal to: avg. We will give an explicit definition of this function in the simulations section.

The Receiver Side

When a receiver receives a signature packet S_l, it verifies the signature of S_l and verifies the authenticity of all the packets that have a path to S_l. After each t seconds, the receiver sends to the source of the stream a quality of reception report including the packet loss ratio during the last t seconds. The algorithm at the receiver side is shown in figure 7.6, and the verification procedure is illustrated in figure 7.7.

```
RECEIVER ALGORITHM
do
   receive packet P.
   if P is not a signature packet then
      buffer P;
      buffer hashes included in P;
   else
      /* P is a signature packet */
      verify(P);
   end
while(true).

upon timeout do
   generate a quality of reception report R
      including the packet loss ratio;
   send R;
   schedule timeout after t seconds;
end.
```

Figure 7.6: The algorithm at a receiver side

```
VERIFY(P)
if P is a signature packet then
   verify the signature of P;
   if the P's signature is valid then
      P is authentic;
      for each hash h_i included in P do
         verify(P_i);
      end
   else
      P is not authentic;
   end
else
   extract the buffered hash h associated to P;
   /* verify P against its buffered hash code h */
   if H(P) = h then
      P is authentic;
      for each hash h_i included in P do
         verify(P_i);
      end
   else
      P is not authentic;
   end
end.
```

Figure 7.7: The recursive verification procedure

7.2 A^2Cast: Adaptive source Authentication protocol for multiCAST streams

We have proposed the A^2Cast protocol: *Adaptive source Authentication protocol for multiCAST streams* [26]; the basic version of H_2A. Similarly to H_2A, A^2Cast relies on EMSS (that uses redundant random hash-chaining) with the addition that it assumes that there exists a means for receivers to communicate to the sender the packet loss ratio in the network (for example by sending

periodic RTCP [123] Receiver Reports). Relying on this receivers' feedback, the source decides on the best *redundancy degree* to use in order to tolerate the actual packet loss ratio in the network. This way, A^2Cast allows not only to save unnecessary authentication information overhead but also to reach the best authentication verification ratio of received packets. The difference between A^2Cast and H_2A resides in the hash-chaining technique: in A^2Cast we apply only the random hash-chaining phase, so there is no systematic chaining of the hash of the current packet to its next packet. This was the basic version of H_2A, and simulations allowed to improve it to the current version.

7.3 Simulations and performance evaluation

We carried out simulations using *Network Simulator* (NS-2)[1] to evaluate the performance of H_2A and compare it with EMSS[104] and A^2Cast [26].

7.3.1 Simulation parameters

In what follows, we consider a bursty packet loss pattern with bursts having an average length equal to 7 [16]. Then, we considered a stream of $20,000$ packets with a signature packet every 500 packets ($f = 500$), and where a packet is *hash-linked* to packets within the scope of 250 packets $(d = 250)$. The value of f has been arbitrary chosen. In reality, the value of f should be chosen depending on the application level tolerance to latencies, the computation power of communicating parties and the available bandwidth. The general rule is: if the parameter f is long, then receivers will experience important latencies before verification but will not have too much signatures to verify, and the reduced number of signatures will not consume a lot of bandwidth. We developed our simplified RTP/RTCP version over NS-2. Receivers send quality of reception reports including the packet loss ratio every $\theta = 20$ seconds. We considered the distribution of packet loss ratio over time shown in figure 7.8. The overall average, packet loss ratio is 26%, but over time, it varies from 5% to 60%. We aim to reduce the bandwidth overhead (redundancy degree) while increasing the verification ratio.

7.3.2 Adaptation of redundancy degree

Recall that periodically, the source analyzes quality of reception reports. Then the source adapts the redundancy degree accordingly using the *adjust_redundancy_degree* function. To develop this function, we run extensive simulations of our hybrid hash-chaining scheme by varying packet loss ratio from 5% to 60%, and we noted for each packet loss ratio the minimum redundancy degree which allows to reach a very high verification probability of received packets (Namely 99%). Figure 7.9 illustrates the results. As we can see, the hybrid scheme minimizes the redundancy

[1]The Network Simulator NS is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

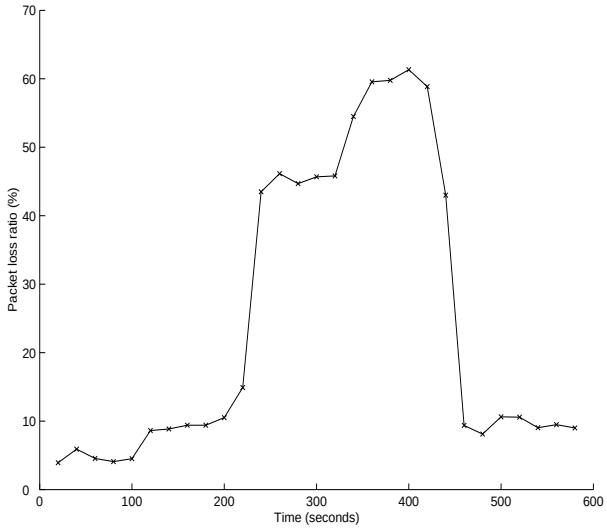


Figure 7.8: The considered scenario of packet loss ratio variation over time

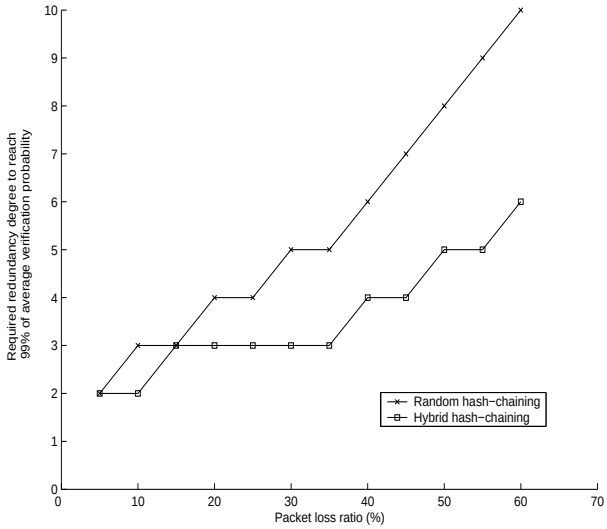


Figure 7.9: Required redundancy degree to reach 99% of verification ratio

degree compared to the only random scheme while maintaining the same performance in term of verification ratio. Hence, given an average loss ratio, our *adjust_redundancy_degree* function returns the minimum redundancy degree which guarantees a very high verification ratio (99%) according to the results of these simulations. In other words, the graph (hybrid hash-chaining) depicted in figure 7.9 corresponds to the *adjust_redundancy_degree* function used by our protocol (H_2A).

7.3.3 Results

We considered a target verification ratio $v = 99\%$, and we run simulations of EMSS, H_2A and A^2Cast to determine the required redundancy degree by each protocol in order to reach the target verification ratio. The results were illuminating: first, the redundancy degree of H_2A over time is obviously proportional to packet loss ratio (compare the shape of the graph representing the redundancy degree of H_2A in figure 7.10 with the shape of the graph representing the variation of packet loss ratio over time in figure 7.8). Besides, we found that H_2A reaches 99% of verification

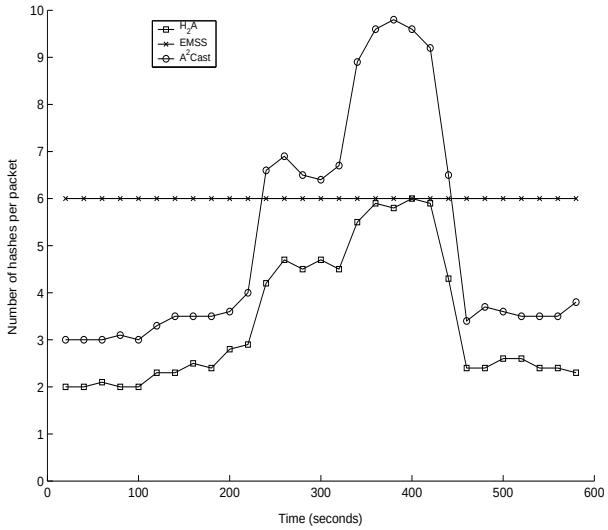


Figure 7.10: The variation of the required redundancy degree to reach 99% of verification ratio

ratio with only an average of 3.35 hashes per packet, whereas EMSS requires 6 hashes per packet to reach the same verification ratio and A^2Cast requires 4.5 hashes per packet (cf. figure 7.11). This means that H_2A allows to save up to 2.65 hashes per packet. If we consider a hash algorithm that produces a 20 byte hash code (such as SHA-1 [49]), this means that H_2A saves up to 1Mbytes of authentication information while sending the $20,000$ packets of the stream. In other words, H_2A allows to save up to 44% of the authentication information used by $EMSS$.

Then we were interested in the impact of the packet loss ratio on the verification ratio of received packets. Figure 7.12 shows that H_2A resists better to packet loss compared to EMSS and A^2Cast. Indeed, when the packet loss ratio reaches 50%, H_2A maintains a verification ratio greater than 98% while EMSS drops to 91% and A^2Cast to 95%. Furthermore, notice that H_2A and A^2Cast use only 3.5 hashes per packet in the average, while EMSS uses 4 hashes per packet. We notice also, that even if EMSS has a greater verification ratio compared to A^2Cast, when the packet loss ratio varies from 10% to 40%, the latter one (A^2Cast) resists better to very high packet loss (50%). Indeed, this is due to the ability of A^2Cast to adapt the *redundancy degree* in order to resist to a such high packet loss ratio

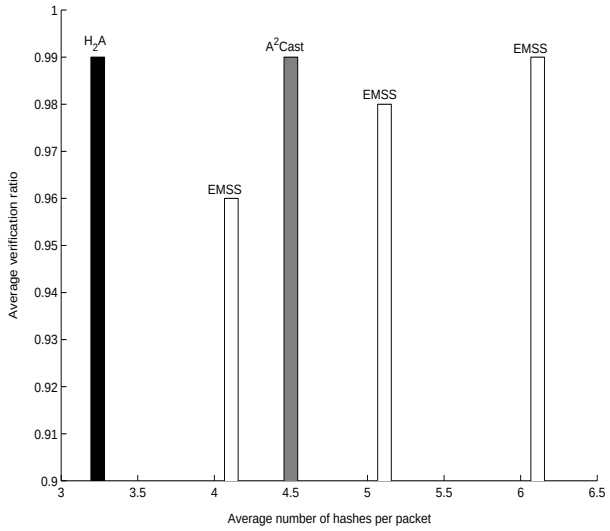


Figure 7.11: Verification efficiency depending on redundancy degree

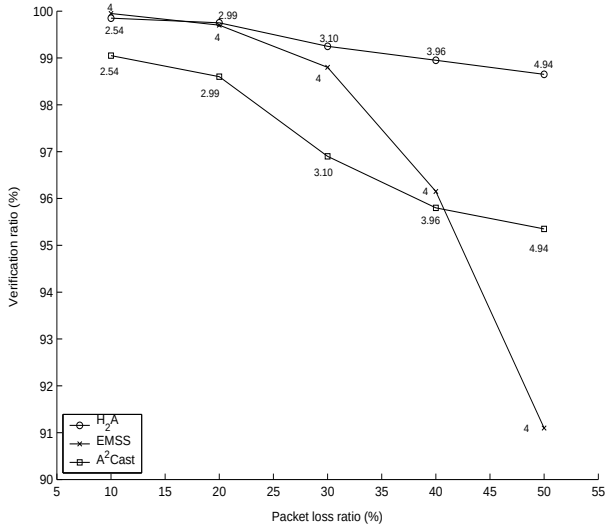


Figure 7.12: Verification efficiency depending on packet loss ratio: the numbers beside the points represent the average *redundancy degree*

7.4 H_2A Security and Performance Comparison

H_2A guarantees data origin authentication and non-repudiation by relying on the existence of *hash-chains* between data packets and *signature packets*. Hence, the security of our protocol (H_2A) relies on the security of this basic technique (hash-chains) which has been proved to be secure by Gennaro and Rohatgi [57]. We have shown in previous sections that H_2A reduces the amount of *authentication information* while maintaining good performance in term of *robustness against packet loss*. However, there are some other performance criteria of H_2A that should be discussed. Table 7.2 compares H_2A to some data origin authentication with non-repudiation pro-

Protocol	Latency at the source	Latency at receivers	Tolerance to packet loss	Authentication information size
Simple Off-line chaining	Yes	No	No	$\|d\|$
EMSS	No	Yes	The authentication probability of a packet is at least 90%	$6\|d\|$
Augmented Chains $C_{a,p}$	Yes	Yes	Yes: Each block of packets tolerates a single burst of length up to $p(a-1)$, where p is the number of packets buffered by the source, and $p+a-1$ is the number of hashes buffered by the source	$2\|d\|$ in average
Piggybacking	Yes	Yes	Yes: Each prioritized packets' set S_i tolerates x_i bursts of b_i packets (x_i and b_i being input parameters)	$\|d\| \times (x_i + 1)$ at least, for a packet in class S_i
H_2A	No	Yes	Yes: 99% average verification ratio	*Source Driven*: depends on average packet loss ratio faced by receivers

$|d|$: size of a digest (hash).

Table 7.2: Comparison of some *multicast data origin authentication with non repudiation* protocols

tocols, described in the related works section, with respect to the following criteria [2]:

1. *The latency at the sender*: corresponds to the fact that the sender needs to buffer packets before sending them.
2. *The latency at a receiver*: corresponds to the fact that a receiver needs to buffer packets before verifying their authenticity.
3. *Tolerance to packet loss*: corresponds to the fact that the authentication process is possible even if some packets are lost.
4. *Authentication information size*: the size of the authentication information embedded to a packet.

H_2A improves multicast data origin authentication performance by saving useless authentication information, and hence reduces the required bandwidth overhead. In what relates to computation overhead, H_2A requires only a single hash computation per packet in addition to a single digital signature computation after each period of f packets. f depends on the maximum delay that the application level can tolerate at the receiver side. Table 7.3 illustrates speed measurement of some famous hash functions when considering different implementations.

	Implement.	MD5	SHA-1
Pentium 4, 2.1 Ghz[35]	Software	204.55 Mbps	72.60 Mbps
FPGA [129]	Hardware	2.1 Gbps	2.3 Gbps

Table 7.3: Speed measurement of Hash functions

[2] With EMSS, we consider results of the special case simulated by authors

In conclusion, simulations show that H_2A adapts well the required authentication information size (redundancy degree) to the actual packet loss ratio in the network and hence allows to reduce the authentication information overhead while maintaining high robustness against packet loss. Since packets can not be verified until the correspondent signature packet is received, receivers experience some delay before verification of received packets. Scalability is not a concern since the number of hash embedding within each packet is independent from the number of multicast group members. H_2A computation overhead is reduced to a single hash computation per packet in addition to a periodic digital signature computation. Besides, H_2A computation efficiency can be further enhanced when considering hardware implementations of the used hash and digital signature algorithms.

7.5 Conclusion

Data origin authentication is a required component in the whole multicast security architecture. Besides, many applications need non-repudiation of data-streams. To achieve non-repudiation, we proposed a new adaptive and efficient protocol called H_2A. Our protocol uses a hybrid and adaptive hash-chaining technique to amortize a single digital signature over many packets. This H_2A's hash-chaining technique allows to save bandwidth and improves the probability that a packet be verifiable even if some packets are lost.

Simulation results that we obtained using NS-2 show that our protocol resists to bursty packet loss and assures with a high probability that a received packet be verifiable. Besides, the simulations and comparisons with other protocols show that our adaptive hash-chaining technique is more efficient than hash-chaining techniques that do not take into consideration the actual packet loss ratio in the network. Indeed, adapting *redundancy degree* to the packet loss ratio allows to save useless authentication information redundancy and hence reduces the bandwidth overhead. Furthermore, the *hybrid hash-chaining* technique allows to maintain high robustness to packet loss.

Chapter 8

Receiver driven Layered Hash-chaining

ONE problem with existing data origin authentication solutions is that they do not take into consideration the distribution of packet loss throughout a large scale network [141]. Indeed, in the existing solutions, the source considers the *worst packet loss ratio* that receivers may encounter in the network and generates the required authentication information redundancy degree to tolerate this *worst case*. This approach assures a high tolerance to packet loss but introduces extra authentication information overhead since it considers the worst case which is likely to appear only at some parts of the network.

In this chapter, we propose an efficient multicast data origin authentication protocol based on a novel layered hash-chaining scheme. We called this protocol: *Receiver driven Layered Hash-chaining for multicast data origin authentication (RLH)*. This protocol tolerates packet loss and guarantees non-repudiation of media-streaming origin. Furthermore, *RLH* allows receivers to make the decision regarding the authentication information redundancy degree depending on the quality of reception in term of packet loss ratio. This novel technique allows to save bandwidth since the packet loss distribution over a large scale network is likely to be not uniform [141]. We have simulated our protocol using NS-2, and the simulation results show that the protocol has remarkable features and efficiency compared to other recent data origin authentication protocols. This chapter is based on [30, 31].

In the following section, we describe our protocol: RLH, then we evaluate and compare it with other protocols using NS-2 simulations.

8.1 RLH: Receiver driven Layered Hash-chaining for multicast data origin authentication

In our protocol, we use signature amortization to assure data origin authentication with non-repudiation for a stream of packets. Besides, we use a *redundant hash-chaining* scheme to tolerate packet loss. The *redundant hash-chaining* that we propose is organized into different layers of redundancy. A basic layer carries the payload data packets in conjunction with a *minimal hash-chaining redundancy degree*. This layer is *vertically* chained to optional layers with different amounts of redundant hash-chains. Each layer is sent to a different multicast group and assures

robustness to a certain amount of packet loss. Periodically, receivers calculate the actual packet loss ratio and use it to decide whether to join a corresponding extra-layer in order to improve the *verification probability*. Figure 8.1 illustrates a scenario where the source produces three layers of authentication information. $L0$ is the compulsory basic layer that carries the payload data packets. $L1$ and $L2$ are authentication information layers that receivers can join to improve the *verification probability*. In this simple scenario, we consider that $L2$ is more redundant than $L1$, and hence $L2$ is joined only by those receivers that encounter a severe packet loss rate in their subnet.

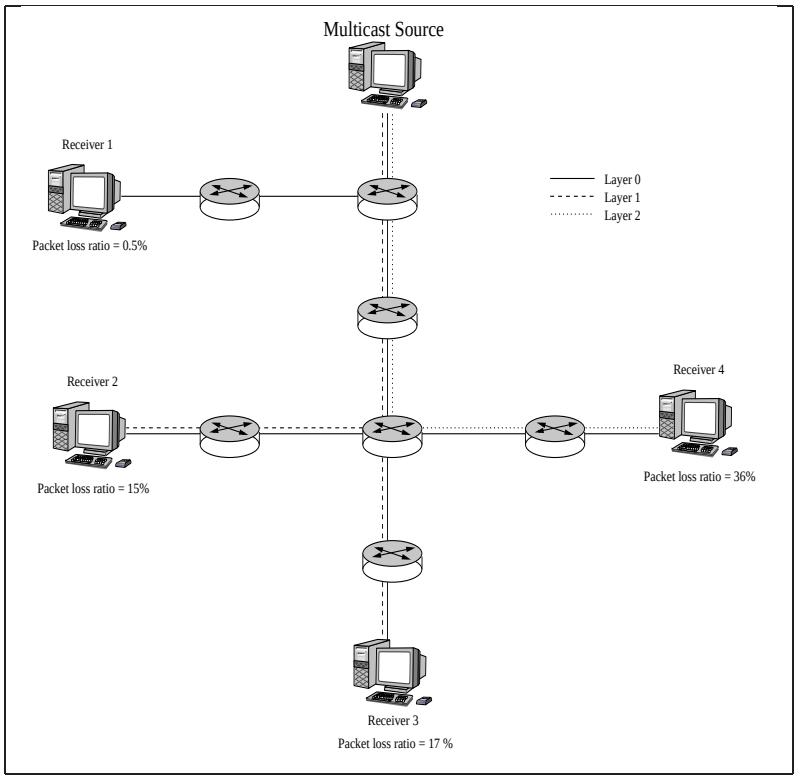


Figure 8.1: A simple RLH scenario with three layers

Since the packet loss distribution over a large scale network is likely to be not uniform [141], this *receiver driven* technique will allow to save bandwidth. Indeed, with this technique, each receiver receives only the required authentication information that allows him to face the actual packet loss ratio in its subnet. In the following paragraphs, we will describe our *layered hash-chaining* scheme. Then we present the RLH protocol.

8.1.1 Layered Hash-chaining scheme

Recall that redundant hash-chaining allows to increase the probability that it remains *hash-link* paths between received packets and signature packets. If a *hash-link* path exists between a received packet and a signature packet, then the authenticity of the received packet is *verifiable* [57, 104]. In our case, we have different layers of redundant hash-chains. The first layer is the basic data payload

layer. It carries data packets chained using a redundant hash-chaining with a *small* redundancy degree. These packets are also chained to other optional layers. Packets of these layers are initially empty. Then, they are hash-chained using different redundancy degrees and hence carry only hashes of packets from the same layer or from the basic layer. It turns out that each layer i is characterized by two redundancy degrees:

1. The *vertical redundancy degree* v_i: determines the number of times the hash of a packet from layer 0 is embedded into packets from layer i.

2. The *horizontal redundancy degree* h_i: determines the number of times the hash of a packet is embedded into subsequent packets from the same layer i.

Figure 8.2 illustrates an example of layered hash-chaining with three layers: the basic layer has horizontal and vertical degrees respectively equal to 2 and 0. Layer 1 has horizontal and vertical degrees respectively equal to 3 and 1, and layer 2 has horizontal and vertical degrees respectively equal to 4 and 1.

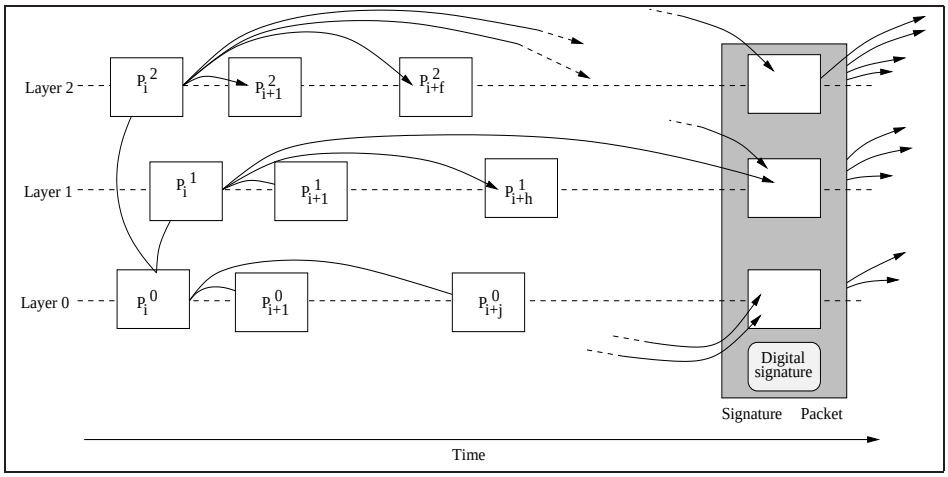


Figure 8.2: Layered Hash-chaining

When a data packet is presented to be sent at the sender, it is *hash-linked* following two steps:

1. *Vertical hash-chaining:* in this step, the hash of the current data packet is embedded within v_i target packets (for each layer i): one packet is the packet that has the same sequence number in layer i and $v_i - 1$ target packets are chosen *randomly*.

2. *Horizontal hash-chaining:* in this step, the hash of the current data packet is embedded into h_0 subsequent *target packets*: one packet is the next one, and $h_0 - 1$ target packets are chosen randomly. Similarly, authentication packets that are beyond the current packet in the other layers are also horizontally chained to h_i subsequent target packets, where i is the layer number. One target packet is the next one and the other $h_i - 1$ target packets are chosen randomly.

(n)	Number of layers
(h_i, v_i)	The horizontal and vertical redundancy degrees of layer i
(f)	Number of packets after which a signature packet is embedded within the stream
(d)	The scope within which packets are chosen randomly to embed the hash of the current packet
(t)	The period of time after which receivers analyze packet loss ratio to decide whether to join a new authentication layer

Table 8.1: RLH parameters

8.1.2 RLH protocol

We consider a multicast source of a stream which consists in a sequence of data packets. The source constructs the different authentication layers according to the layered hash-chaining scheme described above. The source sends each layer i to a different multicast group g_i. In order to assure continuous non-repudiation of the stream, the source embeds within the stream a *signature packet* periodically. As we can see in figure 8.2, a *signature packet* contains:

1. The concatenation of the packet hashes from the different layers for which the signature packet is a *target packet*.
2. A *digital signature* over this hashes' concatenation.

Receivers of the stream join the group g_0 and start verifying the authenticity of received packets relying on the basic redundant hash-chaining of layer 0. Continuously, receivers report lost packets using time-outs and sequence numbers of received packets. Periodically, each receiver uses the packet loss ratio, calculated during the last period of time, to decide whether to join another layer in addition to the basic layer in order to improve the *verification probability*. Indeed, each new layer brings new hash-chains in addition to hash-chains of layer 0, and hence increases the probability that a hash-chain remains between each data packet and a signature packet even if some packets are lost.

Table 8.1 summarizes the parameters involved in RLH protocol. These parameters influence the computation and communication overhead, the delay until verification, and the robustness against packet loss.

The Sender Side Algorithm

In what follows, we denote a packet with a sequence number i and belonging to layer k by P_i^k. A source of a stream applies the layered hash-chaining scheme described above for each packet P_i^0 before it sends it. Packets of layer k are sent to the corresponding multicast group g_k. After each f data packets, the source sends a signature packet. We suppose that signature packets are sent using a certain reliable means, such as retransmission in case of loss using some reliable multicast protocol or by replication that minimizes the probability of loss. The algorithm at the source would then be as shown in figure 8.3.

```
SENDER ALGORITHM
for each packet P_i^0 do
   for each layer k do
      /* make vertical hash-chaining */
      if k ≠ 0 then
         embed H(P_i^0) = h_i^0 in the packet P_i^k;
      do v_k − 1 times
         generate a random number j so that j ∈ [i+1, i+d];
         include H(P_i^0) = h_i^0 in the packet P_{i+j}^k;
      end;

      /* make horizontal hash-chaining */
      embed H(P_i^k) = h_i^k in the packet P_{i+1}^k;
      do h_k − 1 times
         generate a random number j so that j ∈ [i+2, i+d];
         include H(P_i^k) = h_i^k in the packet P_{i+j}^k;
      end;
      send packet P_i^k to multicast group g_k;
   end;
end.

after each f packets do
   sign the current packet S_l;
   send the signature packet S_l to multicast group g_0;
end.
```

Figure 8.3: The algorithm at the source side

The Receiver Side Algorithm

When a receiver receives a signature packet S_l, it verifies the signature of S_l and verifies the authenticity of all the packets that have a path to S_l. Packets that are not authentic and those that do not maintain a chain to a signature packet S_l are labeled by RLH, respectively, *not authentic* and *not verifiable*. The application level uses *authentic* packets, rejects *not authentic* packets and can decide to use *not verifiable* packets depending on the desired security level. After each t seconds, the receiver analyzes the packet loss ratio and decides whether to join another layer to increase *verification probability* of received packets. This decision is made using a function that we call *update_membership* for which it gives the packet loss ratio as a parameter. The algorithm at the receiver side is shown in figure 8.4, and the verification procedure is illustrated in figure 8.5.

8.2 Simulations and performance evaluation

We carried out simulations using Network Simulator v2 (NS-2)[93] to evaluate the performance of *RLH* and compare it with EMSS[104] and A^2Cast[26].

```
RECEIVER ALGORITHM
do
   receive packet P_i^k.
   if P_i^k is not a signature packet then
      buffer P_i^k including the hashes that it carries;
   else
      /* P_i^k is a signature packet */
      verify(P_i^k);
   end;
while(true).

upon timeout do
   update_membership(packet_loss_ratio);
   schedule timeout after t seconds;
end.
```

Figure 8.4: The algorithm at a receiver side

```
VERIFY(P_i^k)
if P_i^k is a signature packet then
   verify the signature of P_i^k;
   if the P_i^k's signature is valid then
      P_i^k is authentic;
      for each hash h_j^m included in P_i^k do
            if P_j^m exists in the received packets' buffer then
               verify(P_j^m);
            end;
      end;
   else
      P_i^k is not authentic;
   end;
else
   /* verify P_i^k against its buffered hash code h_i^k */
   if h_i^k exists in the received hashes' buffer then
      if H(P_i^k) = h_i^k then
         P_i^k is authentic;
         for each hash h_j^m included in P_i^k do
            if P_j^m exists in the received packets' buffer then
               verify(P_j^m);
            end;
         end;
      else
         P_i^k is not authentic;
      end;
   else
      P_i^k is not verifiable;
   end;
end.
```

Figure 8.5: The recursive verification procedure

8.2.1 Simulation parameters

In what follows, we consider a bursty packet loss pattern with bursts having an average length equal to 7. Then, we considered a stream of $10,000$ packets with a signature packet every 500 packets ($f = 500$), and where a packet is *hash-linked* to packets within the scope of 250 packets

$(d = 250)$. The value of f has been arbitrary chosen. In reality, the value of f should be chosen depending on the application level tolerance to latencies, the computation power of communicating parties and the available bandwidth. Receivers analyze packet loss ratio and eventually update their membership to authentication layers every 20 seconds ($t = 20s$).

8.2.2 Updating the membership to authentication layers

Recall that periodically, the receivers analyze the actual packet loss ratio in their subnets. Then use this ratio to join and / or leave authentication layers in order to increase the *verification probability*. This decision is made using the *update_membership* function. To develop this function, we simulated different combinations of different layers with different horizontal and vertical redundancy degrees. At last, we selected the combination of three layers whose verification ratios are illustrated in figure 8.6.

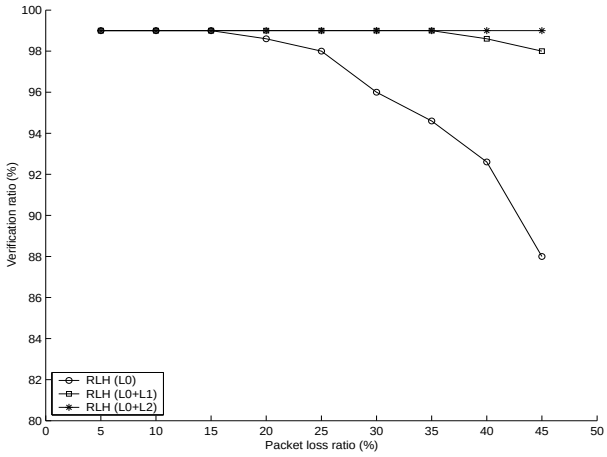


Figure 8.6: The verification ratio of different hash-chain layer combinations

Table 8.2 illustrates the vertical and horizontal redundancy degrees of the selected combination layers.

Layers	Vertical degree	Horizontal degree	Total degree
L0	0	2	2
L0+L1	1	2+3	6
L0+L2	1	2+5	8

Table 8.2: Parameter values of the selected combination of layers

We notice that for packet loss ratios that varies from 5% to 15%, the basic layer suffices to reach 99% of *verification ratio*. The basic layer in addition to layer 1 assure 99% of verification ratio while tolerating up to 35% of packet loss. Finally, the combination of the basic layer with layer 2 assures 99% of verification ratio while tolerating up to 45%. Thus, when a receiver calculates

the encountered packet loss ratio in its subnet, it calls the *update_membership* function depicted in figure 8.7. Without loss in generality, we suppose that the maximum packet loss ratio is 45%

```
function update_membership(loss_ratio)
   join basic layer;
   if 15 <loss_ratio≤ 35 then
      join layer 1;
   if 35 <loss_ratio then
      join layer 2;
end.
```

Figure 8.7: The *update_membership* function

8.2.3 Simulation Results

The adaptive aspect of RLH has been introduced to cope with the variation of packet loss over time and space. Therefore, we will study the behavior of RLH, compared to EMSS and A^2Cast, through three steps: first, we consider the variation of packet loss over time only, then over space only, and finally over both of them simultaneously.

Packet loss variation over time

We consider the distribution of packet loss over time shown in figure 8.8. The packet loss ratio

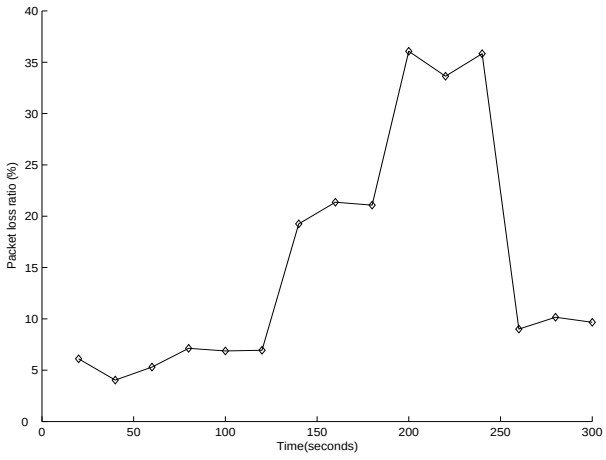


Figure 8.8: Packet loss ratio variation over time

varies, over time, from 5% to 35%. We considered 99% as a target verification ratio, and we run simulations of EMSS, A^2Cast and RLH to determine the required redundancy degree by each protocol in order to reach the target verification ratio. Figure 8.9 illustrates the required redundancy degree by each protocol over time. Notice that the redundancy degrees of RLH and A^2Cast are proportional to packet loss ratio (compare the shape of the graph representing the

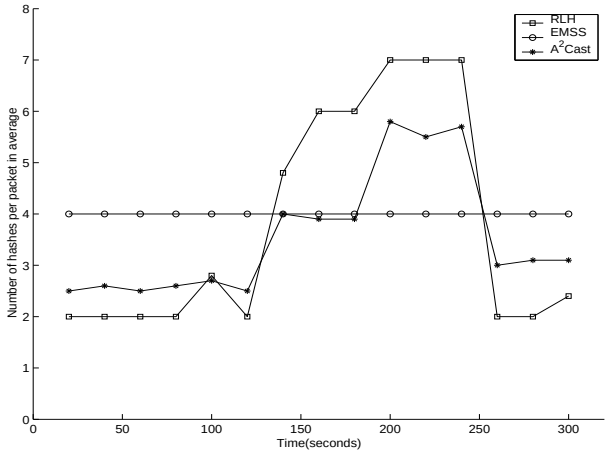


Figure 8.9: The variation of the required redundancy degree to reach 99% of verification ratio

redundancy degrees of RLH and A^2Cast in figure 8.9 with the shape of the graph representing the variation of packet loss over time in figure 8.8). This is due to the fact that RLH adapts the redundancy degree relying on a receiver driven strategy as explained in previous paragraphs. Similarly, A^2Cast adapts the redundancy degree to the actual average packet loss ratio relying on a source driven strategy. On the contrary, EMSS uses a fixed amount of authentication information. Consequently, A^2Cast and RLH allow to save some bandwidth compared to EMSS. For instance, A^2Cast saves up to 88KBytes when considering the overall stream using SHA-1. This saving can be very important when considering large periods of time with high jitters of packet loss ratio. Even though RLH behaves better than EMSS, its benefits are more remarkable in more realistic settings where packet loss distribution varies also over space in addition to its variation over time.

Packet loss variation over space

In order to illustrate the behavior of RLH compared to EMSS and A^2Cast, when considering a large scale network, where the packet loss ratio is likely to be not uniform [141], we considered a network with three different areas. Figure 8.10 illustrates this simplified scenario. Each area is characterized by its own packet loss ratio. Namely, the three areas have respectively 5%, 25% and 45% packet loss ratios .

We want to reach a very high verification ratio (99%). With RLH, each receiver in each area joins the required *hash-chain* layers to reach 99% of verification ratio using the *update_membership* function. In contrast, with EMSS, receivers are not able to choose the best redundancy degree. Figure 8.11 illustrates the required EMSS redundancy degree to reach 99% of verification ratio when we vary the packet loss ratio from 5% to 60%. Therefore, the multicast source has to choose the best redundancy degree so that receivers can verify the authenticity of received packets with a probability equal at least to 99%. Three strategies can be envisioned:

1. *Considering the minimal packet loss ratio*: in this technique, the source considers only the

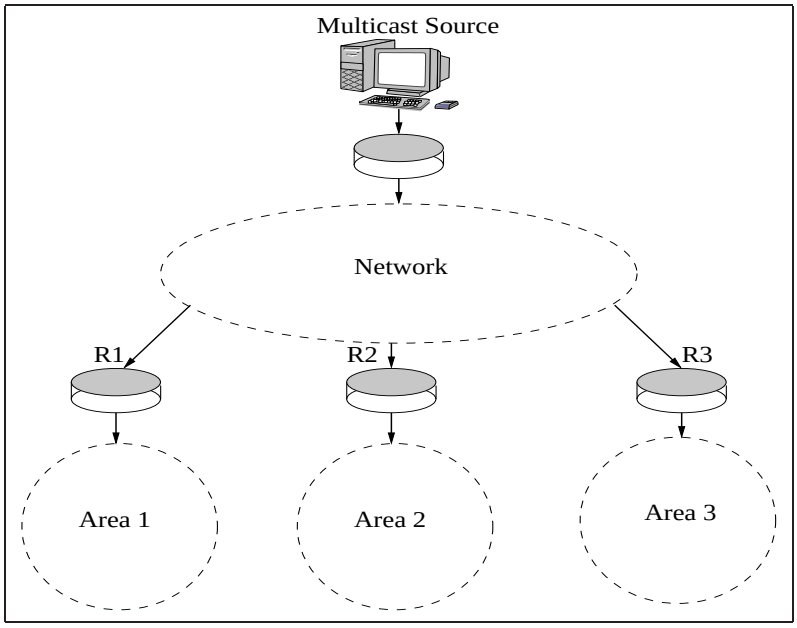


Figure 8.10: Simulation scenario

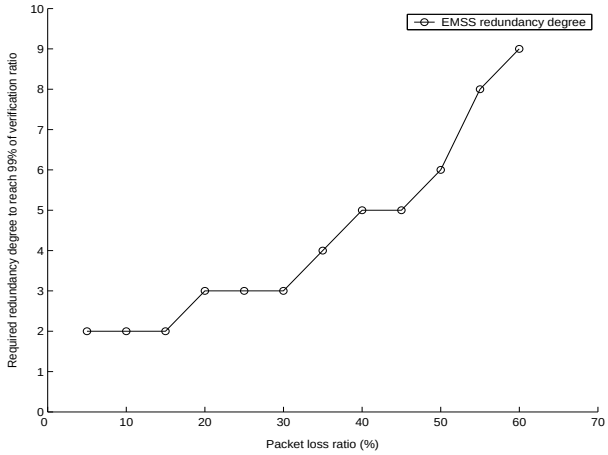


Figure 8.11: The required redundancy degree to reach 99% of verification ratio

area that experiences the minimal packet loss ratio. In this case the source uses the degree 2 which corresponds to the required degree to tolerate 5% of packet loss (cf. figure 8.11). This technique allows to save bandwidth but receivers in the other areas will not reach the 99% verification ratio.

2. *Considering the maximal packet loss ratio*: in this technique, the source considers only the area that experiences the maximal packet loss ratio. In this case the source uses the degree 5 which corresponds to the required degree to tolerate 45% of packet loss (cf. figure 8.11). This technique assures that all receivers in the different areas reach the desired 99% verification ratio, but receivers in areas 1 and 2 will waste bandwidth to receive useless authentication information (extra-redundancy).

3. *Considering the average packet loss ratio*: in this technique, the source considers average packet loss ratio. In this case the source uses the degree 3 which corresponds to the required degree to tolerate 25% of packet loss which is the average packet loss of the three areas (cf. figure 8.11). With this technique, some receivers may not reach the desired verification ratio.

Figure 8.12 illustrates the verification ratio reached within each area using these three different strategies. Notice that none of them achieves the best trade-off between *authentication information bandwidth overhead* and *verification ratio*.

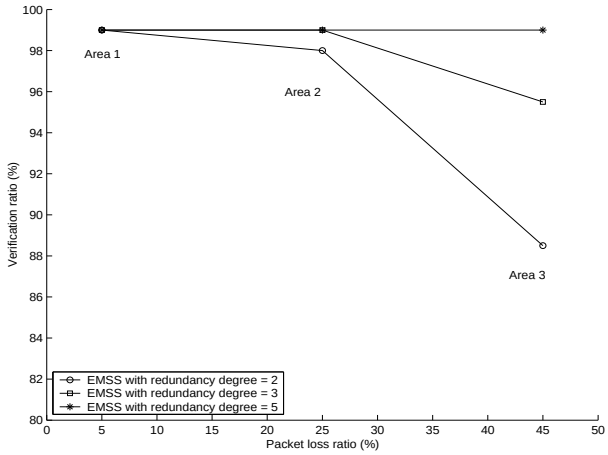


Figure 8.12: The verification ratio within the three areas when considering the three different strategies

In the case of A^2Cast, the source receives periodically quality of reception reports from receivers in the different areas. These reports carry the actual packet loss ratios in each area respectively. Then, the source computes the average packet loss ratio and adjusts the redundancy degree accordingly. Unfortunately, because of relying on the average packet loss ratio, the protocol fails to reach the target verification ratio (99%) for all of those areas. We notice in figure 8.13 that even though A^2Cast saves some bandwidth compared to EMSS and RLH, it failes to reach 99% of verification ratio in areas 2 and 3.

However, in the case of RLH, receivers in area 1 join only the basic layer which suffices to reach the target verification ratio. Receivers of area 2 join the basic layer in addition to layer 1, and receivers of area 3 join the basic layer in addition to layer 2. This way, RLH allows receivers of different areas to save useless bandwidth and to *request* the only required redundancy degree to face the packet loss that is encountered in their respective areas. Figure 8.13 compares RLH to EMSS and A^2Cast regarding the authentication information overhead which consists in the embedded hashes that are used to construct the redundant hash-chains. It illustrates also the verification ratios reached by each protocol within each area. To make this comparison, we calculated the number of hashes (the authentication information overhead) that pass through the *on-tree* multicast border routers of each area: R1, R2 and R3 (cf. figure 8.10).

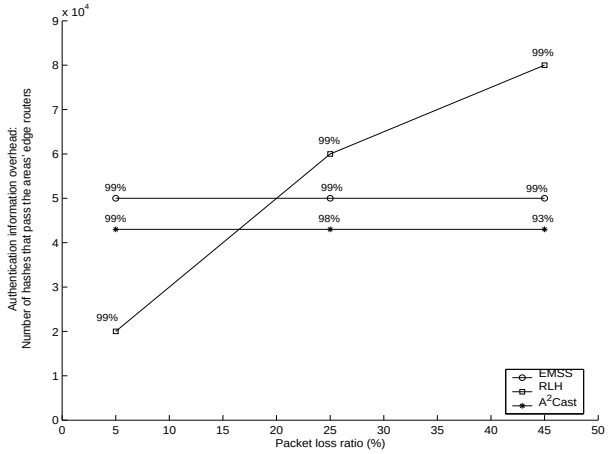


Figure 8.13: The authentication information overhead in the different areas

We notice in figure 8.13, that with EMSS[1] and A^2Cast the three areas receive exactly the same amount of authentication information , even if each area experiences a different amount of packet loss ratio. In contrast, with RLH, each area receives a different amount of authentication information (different layers) depending on the encountered packet loss ratio. Figure 8.14 illustrates the repartition of the *authentication information overhead* per area due to each layer. As expected, receivers of area 1 receive only layer 0 packets. Receivers in area 2 receive layer 0 and layer 1 packets, and receivers in area 3 receive layer 0 and layer 2 packets. This is due to the fact that receivers in each area join only the required layers to reach the target *verification ratio*.

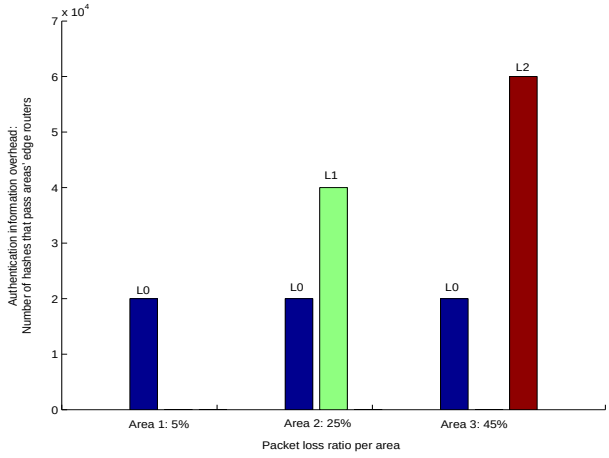


Figure 8.14: The authentication information overhead induced by each layer in the different areas

To further illustrate how RLH allows to save bandwidth, let us consider the second scenario depicted in figure 8.15. The multicast source streams the three RLH layers: the basic data payload layer (layer L0), the medium redundant authentication layer (layer L1), and the highly redundant authentication layer (layer L2). The dashed lines determine the three areas with the different packet

[1]We considered the maximal packet loss ratio strategy so that all receivers reach the target verification ratio

loss ratios. We were interested in measuring the *tree authentication information cost*, which we

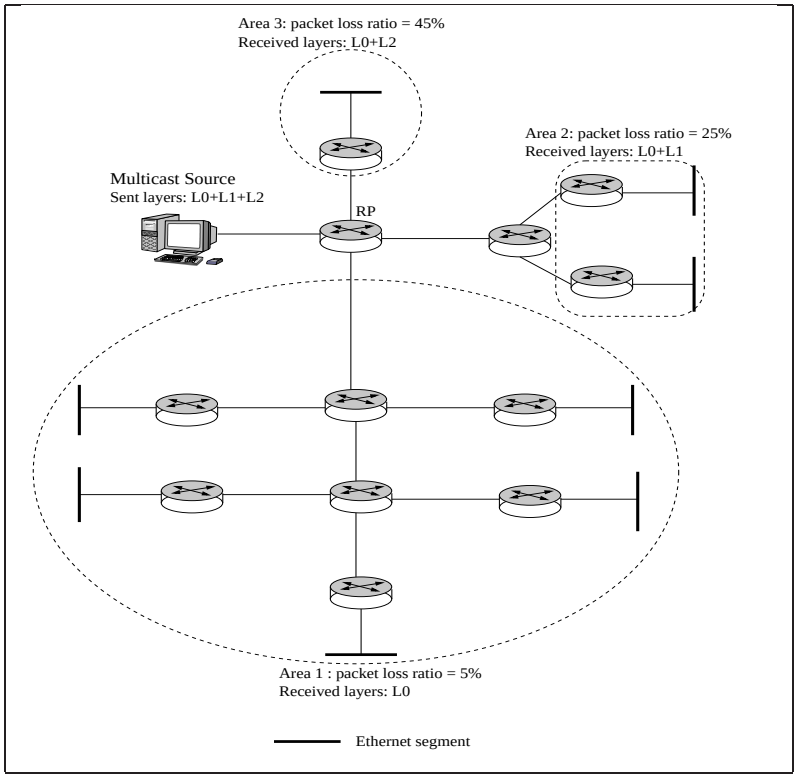


Figure 8.15: Simulation scenario with not uniform packet loss distribution

define as follows:

Definition 7 *The tree authentication information cost is the number of hashes, sent over a multicast tree, by the size of the multicast tree. We mean by the size of a multicast tree the number of network links that constitute the multicast tree. Thus the tree authentication information cost measures the total authentication information bandwidth overhead.*

In our simulation, we used the NS2 implementation of PIM-SM protocol, with RP as a Rendezvous Point node (cf. figure 8.15). In this scenario we considered a $5,000$ packet stream. As we can see in figure 8.16, RLH induces the creation of three multicast groups that correspond to its three layers: L0, L1 and L2. The sizes of these multicast trees are respectively: 12, 4 and 2. Figure 8.16 illustrates the *tree authentication information cost* induced by RLH compared to the ones induced by EMSS and A^2Cast.

With RLH, to each layer corresponds a *tree authentication information cost*: L0 spans all the receivers in the three areas with a *redundancy degree* equal to 2 hashes per packet. L1 spans only receivers of area 2 with a *redundancy degree* equal to 4 hashes per packet, and finally L2 spans only receivers of area 3 with a *redundancy degree* equal to 6 hashes per packet. The three layers induce *tree authentication information costs*, respectively equal to: $104,500$, $65,000$ and $46,500$ hashes in average. In contrast, with EMSS there is a single tree that spans all the receivers

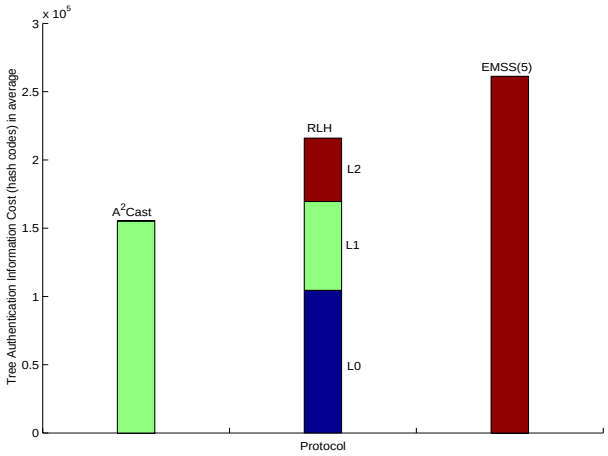


Figure 8.16: Tree authentication information cost

in the three areas with a *redundancy degree* equal to 5, and hence induces a *tree authentication information cost* equal to $261,250$ hashes, in average. Similarly, A^2Cast generates a single tree that spans the three areas with a redundancy degree equal to 2 hashes per packet, inducing a tree authentication information cost equal to $155,289$ hashes. However, since A^2Cast relies on the average packet loss ratio, it fails to reach the 99% verification ratio in the different areas. Namely, it reaches only 32% and 86% of verification ratios in areas 3 and 2, respectively.

According to the results depicted in figure 8.16, we notice that the overall RLH *tree authentication information cost* (sum of the three layer costs) is roughly $50,000$ hashes less than the cost induced by EMSS. If we consider a 160 bit hash function (such as SHA-1), RLH would then save up to 1 MBytes of *tree authentication information*. This is due to the fact that with EMSS, the source considers the maximum redundancy degree so that all receivers reach the same target verification ratio. Whereas, with RLH, receivers join only the required authentication layers to reach the target verification ratio. Therefore, RLH allows receivers to adapt the redundancy degree depending on the actual encountered packet loss ratio, and hence allows to save bandwidth while maintaining high verification probability. In contrast, A^2Cast saves bandwidth but fails to reach high verification probability in the case where packet loss distribution is not uniform over space. This can be easily understood by the fact that the source relies on the average packet loss ratio to adjust the redundancy degree and hence receivers that face an actual packet loss ratio beneath the average will fail to reach high verification probabilities.

Packet loss variation over time and space

In this last scenario, we study the impact of packet loss variation over both time and space on the performance of RLH, A^2Cast and EMSS. We consider a network with three areas, as illustrated in figure 8.17, where each area is characterized by its packet loss pattern as depicted in figure 8.18.

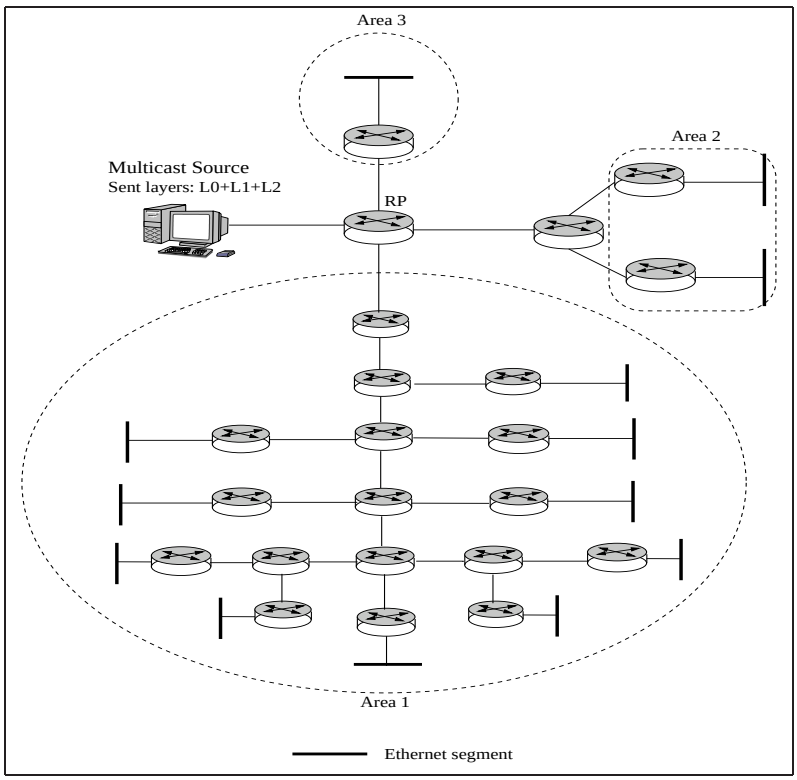


Figure 8.17: Simulation scenario with not uniform packet loss distribution over time and space

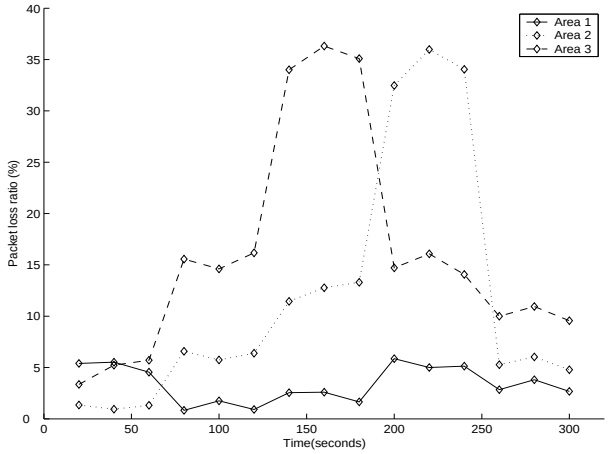


Figure 8.18: Packet loss variation over time within each area

The repartition of group members is also not uniform through the three areas. Namely, there are 10 receivers in area 1, 2 receivers in area 2 and a single receiver in area 3 (cf. figure 8.17). With these settings, figure 8.19 shows the induced redundancy degree by each protocol for each area, and figure 8.20 illustrates the *Tree Authentication Cost* induced by each protocol.

EMSS requires a 4 redundancy degree to reach 99% of verification ratio. With EMSS, the same amount of redundant authentication information reaches the different areas during the whole con-

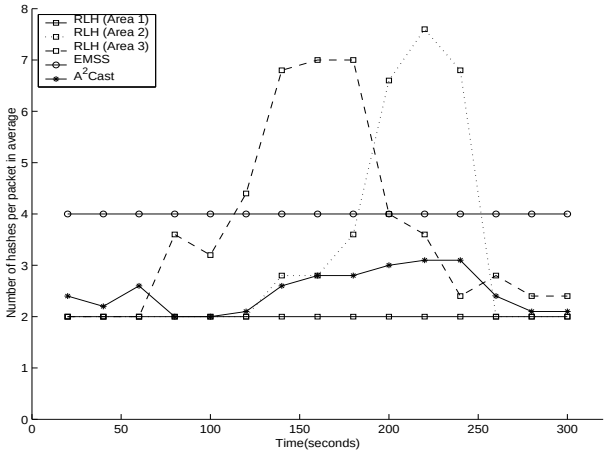


Figure 8.19: Redundancy degree variation over time within each area

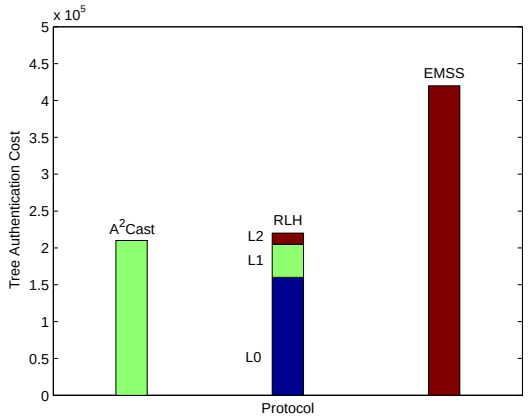


Figure 8.20: Tree authentication information cost

sidered period of time. In contrast, A^2Cast adapts the redundancy degree according to the **average** packet loss ratio calculated at the source. However, this strategy failes to reach the target verification ratio at the overall areas. Indeed, with A^2Cast, receivers at areas 2 and 3 reach only 98% and 95% of verification ratio, respectively. However, with roughly the same *Tree Authentication Cost* as A^2Cast, as depicted in figure 8.20, RLH allows to reach 99% of *verification ratio* at the overall areas. Finally, we notice that RLH allows to adapt the authentication information redundancy depending on the actual packet loss ratio faced by receivers at different locations and at different moments. In fact, we see in figures 8.18 and 8.19 that for each area, characterized by a specific packet loss pattern, corresponds an evolution of the authentication information redundancy that is proportional over time. Hence, RLH allows to reach the highest verification ratio (99% in this case) while minimizing the overall *tree authentication cost*.

8.2.4 RLH security and other performance criteria

RLH guarantees data origin authentication and non-repudiation by relying on the existence of *hash-chains* between data packets and *signature packets*. Hence, the security of our protocol (RLH) relies on the security of this basic technique (hash-chains) which has been proved to be secure by Gennaro and Rohatgi [57]. We have shown in previous sections that RLH reduces the amount of *tree authentication information* while maintaining good performance in term of *robustness against packet loss*. Furthermore, we summarize some other features of RLH in what follows:

1. *Storage requirement and delay before verification at receivers*: with RLH, a receiver experiences a delay before verification of received packets, because it has to receive the signature packet which corresponds to received packets in order to launch the verification process. Hence, receivers need to buffer received packets until the reception of the corresponding signature packet. The duration of the delay and the size of the buffer depend on the period (f packets) after which signature packets are sent.

2. *Storage requirements and delay before authentication at the source*: with RLH, the source authenticates the packets and signs the stream on the fly. Hence the multicast source does not experience any delay before authenticating the stream packets. However, the source has to store packet hashes of the previous packets, from the different layers, until it processes all the packets that are supposed to carry a copy of those hashes. Because of using different authentication layers, RLH source storage requirement is higher than the storage requirements of EMSS and A^2Cast. Figure 8.21 illustrates the buffer size evolution for the three protocols, when considering packet loss variation over time only as depicted in figure 8.8.

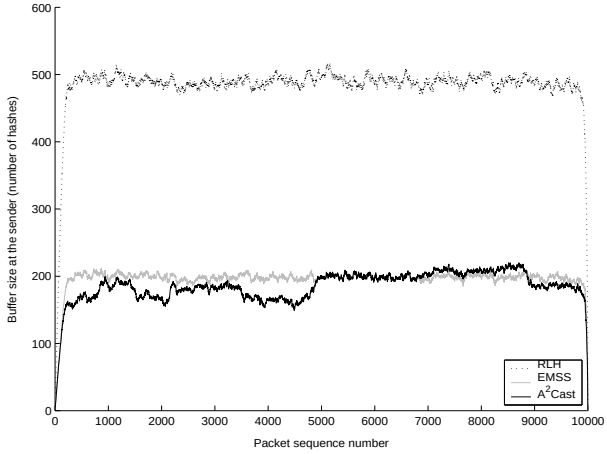


Figure 8.21: Sender Buffer size evolution

In average RLH, EMSS and A^2Cast require to buffer 483, 196 and 185 hashes, respectively. Given effective hash sizes (128 bits for MD5, 160 bits for SHA-1), RLH sender would have

to be able to store 10 KBytes of hashes in average. Obviously, this would not be an issue with current memory sizes.

3. *Processing overhead at the sender*: if we consider a general case of RLH with L layers, the sender would have to calculate L hashes per data packet (one hash per layer), in addition to a digital signature computation after each f consecutive packets. Wei Dai showed in his benchmark [35] that the speed of a MD5 [116] hash calculation is 204.55 Mbps using a Pentium IV, 2.1 Ghz processor. The same processor takes 4.65 ms to calculate a RSA-1024 [117] digital signature. EMSS and A^2Cast induce less computation overhead at the sender, since both of them require only a single hash computation per packet in addition to a single digital signature after each segment of f packets.

4. *Scalability*: since the hash-chaining technique used by RLH is independent from the number of receivers, the protocol scales to large groups.

In conclusion, RLH efficiency increases when the multicast tree size is important and the packet loss phenomenon is concentrated in dense areas. A^2Cast behaves better than RLH when packet loss distribution is roughly uniform over space. Both EMSS and A^2Cast induce less storage and computation overheads at the sender, compared to RLH.

8.2.5 Comparison

Table 8.3 compares RLH with some data origin authentication with non-repudiation protocols, described in the related works section, with respect to the following criteria [2]:

1. *The latency at the sender*: corresponds to the fact that the sender needs to buffer packets before sending them.

2. *The latency at a receiver*: corresponds to the fact that a receiver needs to buffer packets before verifying their authenticity.

3. *Tolerance to packet loss*: corresponds to the fact that the authentication process is possible even if some packets are lost.

4. *Authentication information size*: the size of the authentication information embedded to a packet.

8.3 Conclusion

To achieve non-repudiation, we proposed a new efficient protocol called *RLH*. Our protocol uses a layered hash-chaining technique to amortize a single digital signature over many packets. This

[2] With EMSS, we consider results of the special case simulated by authors

Protocol	**Latency at the source**	**Latency at receivers**	**Tolerance to packet loss**	**Authentication information size**
Simple Off-line chaining	Yes	No	No	$\|d\|$
EMSS	No	Yes	The authentication probability of a packet is at least 90%	$6\|d\|$
p-random graphs	Yes	No	Pr(P_i is verifiable $\mid P_i$ is received)$\geq 1-(1-p)(1-(p(1-q))^2)^{i-2}$	$p(n-1)$ hashes in average
Augmented Chain $C_{a,p}$	Yes	Yes	Yes: Each block of packets tolerates a single burst of length up to $p(a-1)$	$2\|d\|$ in average
Piggybacking	Yes	Yes	Yes: Each prioritized packets' set S_i tolerates x_i bursts of b_i packets	$\|d\| \times (x_i+1)$ at least, for a packet in class S_i
A^2Cast	No	Yes	Yes: 99% average verification ratio	*Source Driven*: depends on average packet loss ratio faced by receivers
RLH	No	Yes	Yes: 99% average verification ratio	*Receiver Driven*: depends on actual packet loss ratio faced by each receiver

$|d|$: size of a digest (hash). n: number of packets in a block. $|S|$: size of a signature. q: loss probability of a packet.

Table 8.3: Comparison of some *data origin authentication with non repudiation* protocols

RLH's hash-chaining technique allows receivers to limit the authentication information bandwidth overhead to only the required overhead that allows to reach a given packet *verification ratio*. Simulation results using NS-2 show that our protocol resists to bursty packet loss and assures with a high probability that a received packet be verifiable. Besides, the simulations and comparisons with two other protocols show that our layered hash-chaining technique allows to save bandwidth since the packet loss phenomenon is likely to be not uniform over a large scale network. The analysis of RLH compared to other protocols showed also that RLH induces more important storage and computation overheads that can be tolerated given the capacities of current processors and memories.

Chapter 9

Conclusions and Future Work

MULTICASTING is a promising communication model for multi-party applications. The evident benefits of multicasting such as saving bandwidth and efficiency are typically interesting for novel multi-party services combining voice, video and text over Internet. This urges the effective large scale deployment of multicasting to satisfy the increasing demand for multicasting from different network operators and service providers. Unfortunately, the strengths of multicasting are also its security weaknesses. Indeed, we showed in this thesis that there are serious conflicts between multicast and security: the anonymous membership based on a single multicast address that makes the openness and efficiency of multicasting, complicates confidentiality which requires individual and explicit identification of members in order to provide them with the right keys to access the encrypted multicast content. Moreover, large scale groups with highly dynamic members present serious scalability issues for *group key management and distribution*. In what relates to authentication, the multi-party nature of multicasting requires the usage of an efficient asymmetric mechanism to provide data origin authentication. Besides, since most of media-streaming applications based on multicasting, rely on best effort channels, those asymmetric authentication mechanisms must tolerate packet loss.

In this thesis, we focused on two main security services for any secure multicast architecture: *confidentiality* and *data origin authentication*. We showed that the distributed nature of multicasting has a huge impact on security efficiency. On one hand, a multicast distribution tree can span large networks where members may be tremendously distant from each other. On the other hand, we showed in this thesis that the efficiency of multicast security mechanisms can be severely affected by some phenomena that depend on their occurrence location in the network. One keystone phenomenon in the efficiency and scalability of group key management is the *dynamism of group members* which is likely to be different from a location to another, and from a moment to another during the whole multicast session. Another determining factor in the performance equation of data origin authentication is *packet loss phenomenon* whose distribution is usually not uniform over both time and space. Given these group communication features, we proposed in this thesis a set of efficient group key management and data origin authentication protocols that take into consideration the distributed nature of multicasting. This awareness allows to reach high levels of scalability and better performance trade-offs. Indeed, simulations demonstrated that the Scalable and Adaptive Key Management protocol (SAKM) scales to large and highly dynamic groups without tremendously affecting data path while mitigating the *1-affects-n* phenomenon. Simulations

showed also that the *source-driven (H_2A)* and *receiver-driven (RLH)* adaptive data origin authentication protocols tackle efficiently the problem of packet loss variation over time and space and thus reduce the bandwidth overhead while maintaining high levels of verification ratio.

In our opinion, the most promising secure multicast architectures are *decentralized proposals* relying on *multi-domains* with *local group controllers*. This reflects the reality of the Internet composed of *Autonomous Systems*, and complies with legal constraints relating to the use of *cryptography* that may be different from a country to another. Moreover, we believe that *auto-configuration* and *auto-adaptation* are the main features of security mechanisms for *distributed* communication models such as *multicasting*. This allows to *tailor* the security mechanisms to the *specific context* of different users that are likely to be in different locations, using different network technologies with heterogeneous equipments and facing miscellaneous *performance challenges*. This *auto-adaptation* allows to optimize the usage of different resources such as bandwidth, computation power, and storage. Moreover, this aids to cope with local challenges of different users without interference or even compromising the quality of the service or the security level provided for other users in the network.

Future directions The recent developments in the domains of wireless communications and pervasive computing incite the different business operators to deploy multicast and broadcast based applications combining voice, video and text (such as: video-conferencing, interactive group games, news and stock quotes feeding, video on demand, ...) over wireless devices equipped with more and more powerful processors (PDAs, Laptops, Cell phones, ...), and in environments without infrastructures (Ad Hoc networks). Besides, the recent advances in the Micro-Electro-Mechanical technologies (MEMS) allowed the development of micro-components that combine sensor capacities and wireless communication facilities into the same circuit, with reduced dimensions and a reasonable cost. These components, commonly called: micro-sensors, motivated the development of wireless sensor networks based on the collaboration of a high number of autonomous micro-nodes that communicate through multi-hops with reduced scopes. Among the domains of application of sensor networks, one can cite: health, military, intelligent houses, environment, industry, etc. However, the large scale deployment of this kind of applications cannot be achieved without securing the multicast model over this type of wireless networks.

Many key problems inherent to the distributed nature of multicasting and to the resource constraints in sensor equipments and in wireless devices, in general, make securing group communication over this type of networks more difficult to tackle. Among other problems, one can cite:

1. *Efficient security mechanisms for resource constrained devices*: most of the proposed solutions in the literature for securing group communications (confidentiality, authentication and non-repudiation, ...) require relatively high computation and storage capacities. However, the devices used in ad hoc and sensor networks, such as: PDAs, laptops, cell phones,..., have relatively limited resources in terms of computation and storage. Appropriate mechanisms have to be developed to assure the same security levels with lower resource requirements.

2. *Absence of third trust party*: existing solutions suppose in general the existence of third trust entities responsible for authentication, authorization and access control. However, ad hoc and sensor networks are without infrastructures. Therefore, it is difficult to assume the existence of such entities. Thus, it is necessary to develop new trust models suitable for networks without infrastructures in hostile environments such as: war battles and devastated areas following natural disasters (earthquake, Tsunami, tornado, ...).

3. *Security Context and Mobility Impact*: in contrary to static environments, mobility in ad hoc networks can have important impacts on security efficiency and performance. Moreover, mobile nodes establish a certain security context that they want to maintain after their movements without having to negotiate it again. Works have to be conducted to tackle mobility issues relating to group communication security.

4. *Masking heterogeneity*: *next generation networks* consist not only in collaborative autonomous systems, but also in heterogeneous technologies that aim to provide the same service for users connected using different sorts of terminals. The challenge is then, how to design portable security mechanisms for theses different technologies, in order to guarantee safety and dependability for end-users in a seamless way.

We believe that some of our ideas introduced in this thesis, such as *automatic adaptation* and *dynamism awareness* would be of a high interest to apply and adapt for this extended context.

Bibliography

[1] A. Adams, J. Nicholas, and W. Siadak. *Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised)*, June 2004. Internet draft, work in progress.

[2] K. Almeroth and M. Ammar. Collecting and modelling the join/leave behaviour of multicast group members in the Mbone. *Symposium on High Performance Distributed Computing*, 1996.

[3] K. Almeroth and M. Ammar. Multicast group behaviour in the internet's multicast backbone (Mbone). *IEEE communications Magazine*, 1997.

[4] D. Balenson, D. McGrew, and A. Sherman. *Key Management for Large Dynamic Groups : One-Way Function Trees and Amortized Initialization*. draft-balenson-groupkeymgmt-oft-00.txt, February 1999. Internet-Draft.

[5] A. Ballardie. *Scalable Multicast Key Distribution*, May 1996. RFC 1949.

[6] A. Ballardie. *Core Based Trees (CBT version 2) Multicast Routing protocol specification*, September 1997. RFC 2189.

[7] T. Ballardie and J. Crowcroft. Multicast-specific Security Threats and Counter-Measures. *Symposium on Network and Distributed System Security*, page 2:16, February 1995.

[8] T. Ballardie, I.P. Francis, and J. Crowcroft. Core Based Trees: an Architecture for Scalable Inter-domain Multicast Routing. *ACM SIGCOMM*, pages 85–95, 1993.

[9] C. Becker and U. Wille. Communication complexity of group key distribution. *5th ACM Conference on Computerand Communications Security*, Nowember 1998.

[10] F. Bergadano, D. Cavagnino, and B. Crispo. Individual Single Source Authentication on the MBone. *IEEE International Conference on Multimedia and Expo*, 2000.

[11] F. Bergadano, D. Cavagnino, and B. Crispo. Individual Authentication in Multiparty Communications. *Computers and Security*, 21(8):719–735, 2002.

[12] H. Bettahar, A. Bouabdallah, and Y. Challal. AKMP: an Adaptive Key Management Protocol for secure multicast. *IEEE-IC^3N: The 11th International Conference on Computer Communication and Networks*, pages 190–195, October 2002.

[13] D. Bleichenbacher and U. Maurer. On the Efficiency of One-time Digital Signatures. *Advances in Cryptography - ASIACRYPT*, pages 145–158, November 1996.

[14] D. Bleichenbacher and U.M. Maurer. Optimal Tree-based One-time Digital Signature Schemes. 13^{th} *Symposium on Theoretical Aspects of Computer Science (STACS'96*, LNCS(1046):363–374, 1996.

[15] D. Boneh, G. Durfee, and M. Franklin. Lower Bounds for Multicast Message Authentication. *Eurocrypt'01*, LNCS(2045):437–452, 2001.

[16] M. Borella, D. Swider, S. Uludag, and G. Brewster. Internet packet loss: Measurement and implications for end-to-end qos. *International Conference on Parallel Processing*, August 1998.

[17] M.S. Bouassida, I. Chrisment, and O. Festor. An Enhanced Hybrid Key Management Protocol for Secure Multicast in Ad Hoc Networks. *IFIP - Networking*, LNCS(3042):725–742, July 2004.

[18] C. Boyd. On key agreement and conference key agreement. *Information Security and Privacy: Australasian Conference*, LNCS(1270):294–302, 1997.

[19] B. Briscoe. MARKS: Multicast key management using arbitrarily revealed key sequences. *1st International Workshop on Networked Group Communication*, November 1999.

[20] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. *EUROCRYP'94*, LNCS(950):275–286, 1994.

[21] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. *Internet Group Management Protocol, Version 3*, October 2002. RFC 3376.

[22] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast Security: A taxonomy and Efficient Constructions. *IEEE INFOCOM*, pages 708–716, March 1999.

[23] G. Chaddoud, I. Chrisment, and A. Shaff. Dynamic Group Communication Security. *6th IEEE Symposium on computers and communication*, 2001.

[24] Y. Challal, H. Bettahar, and A. Bouabdallah. A Scalable and Adaptive Key Management Protocol for Group Communication. *Wired and Wireless Internel Communications (WWIC'04)*, LNCS(2957):260–271, February 2004.

[25] Y. Challal, H. Bettahar, and A. Bouabdallah. A Taxonomy of Multicast Data Origin Authentication: Issues and Solutions. *IEEE Communications Surveys and Tutorials*, 6(3):34–57, 2004.

[26] Y. Challal, H. Bettahar, and A. Bouabdallah. A^2Cast: an Adaptive source Authentication protocol for multiCast streams. *IEEE-ISCC'2004*, June 2004.

[27] Y. Challal, H. Bettahar, and A. Bouabdallah. Hybrid and Adaptive Hash-chaining scheme for data-streaming source authentication. *7th IEEE-High Speed Networks and Multimedia Communication*, July 2004.

[28] Y. Challal, H. Bettahar, and A. Bouabdallah. SAKM: A Scalable and Adaptive Key Management Approach for Multicast Communications. *ACM SIGCOMM Computer Communications Review*, 34(2):55–70, April 2004.

[29] Y. Challal, A. Bouabdallah, and H. Bettahar. H_2A: Hybrid Hash-chaining scheme for Adaptive multicast source authentication of media-streaming. *Computers & Security Journal*, 24(1):57–68, February 2005.

[30] Y. Challal, A. Bouabdallah, and Y. Hinard. Efficient multicast source authentication using layered hash-chaining scheme. *IEEE Local Computer Networks, LCN'04*, pages 411–412, 2004.

[31] Y. Challal, A. Bouabdallah, and Y. Hinard. RLH: receiver driven layered hash-chaining for multicast data origin authentication. *Computer Communications Journal*, to appear.

[32] K.C. Chan and S.H.G. Chan. Distributed Servers Approach for Large-Scale Secure Multicast. *The IEEE Journal On Selected Areas in Communications*, 20(8):1500:1510, October 2002.

[33] G. H. Chiou and W. T. Chen. Secure Broadcast using Secure Lock. *IEEE Transactions on Software Engineering*, 15(8):929–934, August 1989.

[34] H.H. Chu, L. Qiao, and K. Nahrstedt. A Secure Multicast Protocol with Copyright Protection. *ACM SIGCOMM Computer Communications Review*, 32(2):42:60, April 2002.

[35] W. Dai. *Comparison of popular cryptographic algorithms*. http://www.eskimo.com/w̃eidai/benchmarks.html, 2003.

[36] B. DeCleene, L. Dondeti, S. Griffin, T. Hardjono, D. Kiwior, J. Kurose, D. Towsley, S. Vasudevan, and C. Zhang. Secure group communications for wireless networks. *MILCOM*, June 2001.

[37] S. Deering. *Host Extensions for IP Multicasting*, August 1989. RFC 1112.

[38] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The pim architecture for wide-area multicast routing. *ACM Transactions on Networks*, April 1996.

[39] S. Deering, W. Fenner, and B. Haberman. *Multicast Listener Discovery (MLD) for IPv6*, October 1999. RFC 2710.

[40] S.E. Deering. Multicast Routing in Internetworks and Extended LANs. *ACM SIGCOMM*, August 1988.

[41] Y. Desmedt, Y. Frankel, and M. Yung. Multi-receiver / Multi-sender Network Security: Efficient Authenticated Multicast / Feedback. *IEEE INFOCOM'92*, pages 2045–2054, 1992.

[42] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, November 1976.

[43] L. Dondeti, S. Mukherjee, and A. Samal. A distributed group key management scheme for secure many-to-many communication. *Technical Report PINTL-TR-207-99*, 1999.

[44] L. R. Dondeti, S. Mukherjee, and A. Samal. Scalable secure one-to-many group communication using dual encryption. *Computer Communications*, 23(17):1681–1701, November 2000.

[45] L.R. Dondeti, S. Mukherjee, and A. Samal. Comparison of Hierarchical Key Distribution Schemes. *IEEE Globcom Global Internet Symposium*, 1999.

[46] L.R. Dondeti, S. Mukherjee, and A. Samal. *Survey and Comparison of Secure Group Communication Protocols*, 1999. Technical Report.

[47] T. Dunigan and C. Cao. Group Key Management. *Technical Report ORNL/TM-13470*, 1998.

[48] D.X.Shaw. A Note on the Tree Partitioning Problem. *Technical Report*, July 1996.

[49] D. Eastlake and P. Jones. *US Secure Hash Algorithm 1 (SHA1)*, September 2001. RFC 3174.

[50] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification*, June 1998. RFC 2117.

[51] S. Even, O. Goldreich, and S. Micali. On-line/Off-line Digital Signatures. *Advances in Cryptology - Crypto'89*, LNCS(435):263–275, 1990.

[52] S. Even, O. Goldreich, and S. Micali. On-line/Off-line Digital Signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[53] W. Fenner. *Internet Group Management Protocol, version 2*, November 1997. RFC 2236.

[54] A. Fiat and M. Naor. Broadcast Encryption. *CRYPTO'93*, LNCS(773):480–491, 1993.

[55] H. Fujii, W. Kachen, and K. Kurosawa. Combinatorial Bounds and Design of Broadcast Authentication. *IEICE Trans.*, E79-A(4):502–506, 1996.

[56] R. Gennaro and P. Rohatgi. How to Sign Digital Streams. *Advances in Cryptology, CRYPTO'97*, 1997.

[57] R. Gennaro and P. Rohatgi. How to Sign Digital Streams. *Information and Computation*, 165(1):100–116, February 2001.

[58] P. Golle and N. Modadugu. Authenticating Streamed Data in the Presence of Random Packet Loss. *NDSS'01: The Network and Distributed System Security Symposium*, 2001.

[59] L. Gong and N. Shacham. Trade-offs in Routing Private Multicast Traffic. *GLOBECOM'95*, November 1995.

[60] N.M. Haller. The S/Key(tm) One-time password System. *ISOC Symposium on Network and Distributed Security*, February 1994.

[61] T. Hardjono, M. Baugher, and H. Harney. Key Management and Multicast Security: a Survey. *10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2001)*, pages 223–228, June 2001.

[62] T. Hardjono, B. Cain, and I. Monga. Intra-domain Group Key Management for Multicast Security. *IETF Internet draft*, September 2000.

[63] T. Hardjono and L. Dondeti. *Multicast and Group Security*. Artech House, 2003.

[64] T. Hardjono and G. Tsudik. IP Multicast Security : Issues and Directions. *Annales de telecom*, 2000.

[65] T. Hardjono and B. Weis. *The Multicast Group Security Architecture*, March 2004. RFC 3740.

[66] H. Harney and C. Muckenhirn. *Group Key Management Protocol (GKMP) Architecture*, July 1997. RFC 2093.

[67] H. Harney and C. Muckenhirn. *Group Key Management Protocol (GKMP) Specification*, July 1997. RFC 2094.

[68] I. Ingemarson, D. Tang, and C. Wong. A Conference Key Distribution System. *IEEE Transactions on Information Theory*, 28(5):714–720, September 1982.

[69] P. Judge and M. Ammar. Security Issues and Solutions in Multicast Content Distribution: A Survey. *IEEE Network*, pages 30–36, January/February 2003.

[70] B. Kaliski. *The MD2 Message-Digest Algorithm*, April 1992. RFC 1319.

[71] C. Kaufman, R. Perlman, and M. Speciner. *Network Security : Private Communication in a Public World*. Printice Hall Series in Computer Networking and Distributed Systems, 2002.

[72] M. Kellil, I. Romdhani, H.Y. Lach, A. Bouabdallah, and H. Bettahar. Multicast Receiver and Sender Access Control and its Applicability to Mobile IP Environments: A Survey. *IEEE Communications Surveys and Tutorials*, 7(2), 2005.

[73] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant Key Agreement for Dynamic Collaborative groups. *7th ACM Conference on Computer and Communications Security*, pages 235–244, November 2000.

[74] Y. Kim, A. Perrig, and G. Tsudik. Communication-Efficient group Key Agreement. *IFIP SEC*, June 2001.

[75] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, January 1975.

[76] H. Krawczyk, M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*, February 1997. RFC 2104.

[77] P.S. Kruus. A survey of multicast security issues and architectures. *21st NISSC*, 1998.

[78] K. Kurosawa and S. Obana. Characterization of (k,n) multi-receiver authentication. *Information Security and Privacy, ACISP'97*, LNCS(1270):204–215, 1997.

[79] L. Lamport. Constructing digital signatures from a one-way function. *Technical Report SRI international. CLS 98*, October 1979.

[80] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770:772, November 1981.

[81] A. Lysyanskaya, R. Tamassia, and N. Triandopoulos. Multicast Authentication in Fully Adversial Networks. *IEEE Symposium on Security and Privacy*, page 241, May 2004.

[82] D.A. McGrew and A.T. Sherman. Key Establishement in Large Dynamic Groups using One-way Function Trees. *Technical Report TR-0755*, May 1998.

[83] J.A. Menezes, C.V.O. Paul, and S.A. Vanstone. *Hand Book of Applied Cryptography*. 1996.

[84] R. Merkle. A Digital Signature Based on a Conventional Encryption Function. *CRYPTO'87*, LNCS(293):369–378, 1987.

[85] R. Merkle. A Certified Digital Signature. *CRYPTO'89*, LNCS(435):218–238, 1989.

[86] S. Miner and J. Staddon. Graph-Based Authentication of Digital Streams. *IEEE Symposium on Security and Privacy*, 2001.

[87] S. Mittra. Iolus : A Framework for Scalable Secure Multicasting. *ACM SIGCOMM*, 1997.

[88] M. Mitzenmacher and A. Perrig. Bounds and Improvements for BiBa Signature Schemes. *Technical Report (TR-02-02), Harvard University*, 2002.

[89] R. Molva and A. Pannetrat. Scalable Multicast Security in dynamic groups. *6th ACM Conference on Computer and Communication Security*, November 1999.

[90] J. Moy. *Multicast routing extension for OSPF*, March 1994. RFC 1584.

[91] R. Mukherjee and J.W. Atwood. Proxy Encryptions for Secure Multicast Key Management. *IEEE Local Computer Networks - LCN'03*, October 2003.

[92] R. Mukherjee and J.W. Atwood. SIM-KM: Scalable Infrastructure for Multicast Key Management. *IEEE Local Computer Networks - LCN'04*, pages 335–342, November 2004.

[93] NS-2. *The Network Simulator NS-2*. http://www.isi.edu/nsnam/ns/.

[94] S. Obana and K. Kurosawa. Bounds and Combinatorial Structure of (k,n) Multi-receiver A-codes. *Designs, Codes and Cryptography*, 22(1):47–63, 2001.

[95] R. Oppliger and A. Albanese. Distributed registration and key distribution (DiRK). *Proceedings of the 12th International Conference on Information Security IFIP SEC'96*, 1996.

[96] A. Pannetrat and R. Molva. Authenticating Real Time Packet Streams and Multicasts. 7^{th} *International Symposium on Computers and Communications, ISCC'02*, pages 490–495, July 2002.

[97] A. Pannetrat and R. Molva. Efficient Multicast Packet Authentication. *10th Annual Network and Distributed System Security Symposium*, February 2003.

[98] J.M. Park, E.K.P. Chong, and H.J. Siegel. Efficient Multicast Packet Authentication Using Signature Amortization. *IEEE Symposium on Security and Privacy*, 2002.

[99] J.M. Park, E.K.P. Chong, and H.J. Siegel. Efficient Multicast Stream Authentication Using Erasure Codes. *ACM Transactions on Information and System Security*, 6(2):258–285, May 2003.

[100] V. Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[101] A. Perrig. Efficient Collaborative key Management protocols for Secure Autonomous Group Communication. *International Workshop on Cryptographic techniques and E-commerce*, 1999.

[102] A. Perrig. The BiBa One-Time Signature and Broadcast Authentication Protocol. *The 8th ACM Conference on Computer and Communications Security*, November 2001.

[103] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and Secure Source Authentication for Multicast. 8^{th} *Annual Internet Society Symposium on Network and Distributed System Security*, 2001.

[104] A. Perrig, R. Canetti, J.D. Tygar, and D. Song. Efficient Authentication and Signing of Multicast Streams over Lossy Channels. *IEEE Symposium on Security and Privacy*, 2000.

[105] A. Perrig, R. Canetti, J.D. Tygar, and D. Song. The TESLA Broadcast Authentication Protocol. *RSA CryptoBytes*, 5, Summer 2002.

[106] A. Perrig, D. Song, and J.D. Tygar. ELK, a new protocol for Efficient Large-group Key distribution. *IEEE Security and Priavcy Symposium*, May 2001.

[107] A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, and D.E. Culler. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 8:521–534, 2002.

[108] R. Poovendram, S. Ahmed, S. Corson, and J. Baras. A Scalable Extension of Group Key Management Protocol. *2nd Annual ATRIP Conference*, pages 187–191, February 1998.

[109] Federal Information Processing Standards Publication. *Data Encryption Standard (DES)*, December 1993. FIPS PUB 46.

[110] Federal Information Processing Standards Publication. *Digital Signature Standard (DSS)*, May 1994. FIPS PUB 186.

[111] Federal Information Processing Standards Publication. *Advanced Encryption Standard (AES)*, November 2001. FIPS PUB 197.

[112] M. O. Rabin. Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance. *Journal of the Association for Computing Machinery*, 36(2):335–348, April 1989.

[113] S. Rafaeli and D. Hutchison. Hydra: a decentralized group key management. *11th IEEE International WETICE: Enterprise Security Workshop*, June 2002.

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

[115] L. Reyzin and N. Reyzin. Better than BiBa: Short One-time Signatures with Fast Signing and Verifying. 7^{th} *Australian Conference on Information Security and Privacy*, LNCS(2384):144–153, 2002.

[116] R. Rivest. *The MD5 Message-Digest Algorithm*, April 1992. RFC 1321.

[117] R.L. Rivest, A. Shamir, and L.M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[118] O. Rodeh, K. Birman, and D. Dolev. Optimized group rekey for group communication systems. *Network and Distributed System Security*, February 2000.

[119] P. Rohatgi. A Compact and Fast Hybrid Signature Scheme for Multicast Packet Authentication. 6^{th} *ACM Conference on Computer and Communications Security CCS'99*, pages 93–100, November 1999.

[120] R. Safavi-Naini and H. Wang. New Results on Multi-receiver Authentication Codes. *Advances in Cryptology: EUROCRYPT'98*, LNCS(1403):527–541, 1998.

[121] R. Safavi-Naini and H. Wang. Multireceiver Authentication Codes: Models, Bounds, Constructions, and Extensions. *Information and Computation*, 151:148–172, 1999.

[122] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Second edition, 1996.

[123] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, July 2003. RFC 3550.

[124] H. Seba, A. Bouabdallah, N. Badache, H. Bettahar, and D. Tandjaoui. Key Management and Multicast Security: a Survey. *Annales des Telecommunications*, 58(7-8):1090–1129, 2003.

[125] S. Setia, S. Koussih, S. Jajodia, and E. Harder. Kronos: A scalable group re-keying approach for secure multicast. *IEEE Symposium on Security and Privacy*, May 2000.

[126] C. Shields and J.J. Garcia-Luna-Aceves. The Ordered Core Based Tree Protocol. *IEEE INFOCOM'97*, April 1997.

[127] C. Shields and J.J. Garcia-Luna-Aceves. KHIP-A scalable protocol for secure multicast routing. *ACM SIGCOMM Computer Communication Review*, 29(4):53–64, October 1999.

[128] R. Shirey. *Internet Security Glossary*, May 2000. RFC 2828.

[129] N. Sklavos, E. Alexopoulos, and O. Koufopavlou. Networking Data Integrity: High Speed Architectures and Hardware Implementations. *The International Arab Journal of Information Technology*, 1(0):54–59, 2003.

[130] D. Steer, L.L. Strawczynski, W. Diffie, and M. Weiner. A Secure Audio Teleconference System. *CRYPTO'88*, 1988.

[131] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. *3rd ACM Conference on Computer and Communications Security*, pages 31–37, March 1996.

[132] R. Vida and L. Costa. *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*, June 2004. RFC 3810.

[133] D. Waitzman, C. Partridge, and S. Deering. *Distance Vector Multicast Routing Protocol*, November 1988. RFC 1075.

[134] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, , and B. Plattner. The VersaKey Framework : Versatile Group Key Management. *IEEE Journal on Selected Areas in Communications (Special Issues on Middleware)*, 17(8):1614–1631, August 1999.

[135] D. Wallner, E. Harder, and R. Agee. *Key Management for Multicast : Issues and Architecture*. National Security Agency, June 1999. RFC 2627.

[136] C. K. Wong, M. Gouda, and S. S. Lam. Secure Group Communications Using Key Graphs. *ACM SIGCOMM*, 1998.

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

[138] Ch.K. Wong and S.S. Lam. Digital Signatures for Flows and Multicasts. *IEEE/ACM Transactions on Networking*, 7(4), August 1999.

[139] C.K. Wong and S.S. Lam. Digital Signatures for Flows and Multicasts. *IEEE ICNP'98*, October 1998.

[140] C.K. Wong and S.S. Lam. Keystone: A group Key Management Service. *International Conference on Telecommunication*, May 2000.

[141] M. Yajnik, J. Kurose, and D. Towsley. Packet Loss Correlation in the MBone Multicast Network. *IEEE Global Telecommunications Conference (IEEE/GLOBECOM'96)*, pages 94–99, November 1996.

[142] M. Yajnik, S. Moon, J. Kurose, and D. Towsley. Measurement and Modeling of the Temporal Dependence in Packet Loss. *INFOCOM'99*, pages 345–352, March 1999.

[143] Y.R. Yang, X.S. Li, X.B. Zhang, and S.S. Lam. Reliable Group Rekeying: A Performance Analysis. *TR-01-21*, June 2001.