UNIVERSITÁ DEGLI STUDI DI:
Torino

DIPARTIMENTO DI:
Informatica

DOTTORATO DI RICERCA IN: **Informatica**

CICLO:
XVIII

TITOLO DELLA TESI:

Security, privacy and authentication in shared access to restricted data

TESI PRESENTATA DA:
Paolo Dal Checco

TUTORS:
Prof. Francesco Bergadano
Dr. Davide Cavagnino

COORDINATORE DEL CICLO:
Prof. Piero Torasso

ANNI ACCADEMICI:
2002/2003, 2003/2004, 2004/2005

SETTORE SCIENTIFICO-DISCIPLINARE DI AFFERENZA: INF/01

Contents

Contents iii
List of figures vii
Abstract ix
Acknowledgements xi
Chapter 1 Introduction 1
Chapter 2 State of the art 5
2.1 Introduction 5
2.1.1 Systems using dedicated hardware 6
2.1.2 Systems based on cryptographic functions 6
2.2 Hash functions and hash chains 9
2.3 Symmetric key encryption 10
2.4 Public key cryptography 12
2.5 Secret sharing 13
2.6 Group signatures 15
Chapter 3 Secure logging for irrefutable administration 19
3.1 Introduction 19
3.2 Preliminary considerations 20
3.3 Possible approaches 21
3.4 Encryption and exclusion/elusion 24
3.5 Group auditing 26
3.6 Group access to a log line 26
3.7 Observations on multiple groups 28
Chapter 4 Group signatures 31

4.1 Description of group signatures 31

4.2 Definitions 32

4.3 A new solution for group signatures based on standard signatures 34
4.3.1 Introduction 34
4.3.2 Description 34
4.3.3 Elements of the system 35
4.3.4 Communication between components 37
4.3.5 Operations of the system 38
4.3.6 Validation/verification of system operations 41
4.3.7 Benefits and drawbacks 41
4.3.8 Variant of the first solution 43
4.3.8.1 Pseudonym certificates 43
4.3.8.2 Properties 46
4.3.8.3 Comparison of the variant to the original method 48

4.4 A new solution for group signatures based on one-way accumulators 50
4.4.1 Introduction 50
4.4.2 Description 50
4.4.3 Proposed Group Signature Scheme 51
4.4.3.1 System bootstrap 51
4.4.3.2 Signing by A_i (SIGN) 52
4.4.3.3 Verification by anyone (VERIFY) 52
4.4.3.4 Identification of signer (OPEN) 53
4.4.3.5 Member addition 53
4.4.3.6 A possible improvement: incremental group creation 54
4.4.3.7 Adding more keys for a group member 55
4.4.3.8 Properties 55
4.4.4 Adding revocation to the current solution 55
4.4.4.1 System bootstrap 56
4.4.4.2 Adding a member to the group 57
4.4.4.3 Signing of message M by A_i (SIGN) 58
4.4.4.4 Adding more keys for a group member 58
4.4.4.5 Revoking keys 58
4.4.4.6 Verification by anyone (VERIFY) 60
4.4.4.7 Identification of signer (OPEN) 60
4.4.5 Observations and possible improvements 60
4.4.5.1 Observation I 61
4.4.5.2 Example of attack 61
4.4.5.3 Observation II 63
4.4.5.4 Observation III 64
4.4.5.5 New JOIN 64

Chapter 5 Implementation 67

5.1 Introduction 67
5.1.1 Assolo 68

5.2 Software architecture of the system 70

5.3 Communication between Assolo and Log Manager 71

5.4 Selected algorithms 74

5.5 Cryptographic libraries: Crypto++ 76

5.6 Log Manager functionalities 77

5.6.1 Session storage 77

5.6.2 System initialization 80

5.6.3 System stop 81

5.7 DBMS data storage 81

5.7.1 Description of choices 81

5.7.2 Relational schema 82

5.7.3 Table details 83

5.8 System administration tools 88

5.8.1 Auditors anonymous access management 89

5.9 Group access management 90

5.10 Group members management 91

5.11 Data integrity and authenticity 92

5.12 DBMS access management 93

Chapter 6 Conclusion and future work 97

6.1 Summing up the results 97

6.2 Irrefutable administration and its ethical issues 98

6.3 Future work 99

Chapter 7 Bibliography 101

List of figures

Figure 2-1: The polynomial Q.....14
Figure 2-2: Evaluations of the polynomial Q.....15
Figure 3-1: Common share of different polynomials.....30
Figure 4-1: Interaction between entities.....37
Figure 4-2: Schema of the interaction between components.....45
Figure 5-1: Sniffer-based architecture.....68
Figure 5-2: Gateway-based architecture.....68
Figure 5-3: Protocols stack.....69
Figure 5-4: System architecture.....71
Figure 5-5: Protocols stack pointing out what is stored.....74
Figure 5-6: Relational schema of the DBMS tables.....83
Figure 5-7: Privileges of the users on the single tables.....96

Abstract

This thesis presents the results of the research on the fields of security, privacy and authentication concerning specifically shared access to restricted data. Those fields are strictly related, as we will see in the following, to what we call *irrefutable administration* and its background. It is an interdisciplinary work, bringing together concepts belonging to research areas such as Computer Security, Network Security, Intrusion Detection Systems, Group Signatures and Secret Sharing.

The work consists of four parts. In the first part (**Chapter 1** and **Chapter 2**) we introduce the subject of the research and we present the state of the art related to our work. Each section here is related to a different technology underlying the present research, summarizing the main results related to the concepts our study is based on. This will not be a mere summing up, though, because we will try to draw attention to the relationships of the studied concepts with their corresponding use we've made of in the research.

The second part, presented in **Chapter 3**, studies the concepts of *secure logging* related to what we call *irrefutable administration*, focusing on the research we've carried on. The concept is presented keeping in mind that the possible application is that of securely archiving log files related to systems administration with elusion/exclusion properties, enforced anonymity, privacy and security.

As for the third part of the work, as mentioned before the research topics cover more than just one underlying technology, but the most important one concerns *group signatures*. In order to provide a functional framework where *irrefutable administration* can be built, we designed two solutions for group signatures, described in details in the third part of the thesis (**Chapter 4**). Those two solutions are based on different technologies and theoretical principles and suited to different needs of a complex system.

In the third part, **Chapter 5**, we present the implementation we've developed of a complex system for irrefutable administration, based on the solution described in the second part of this work (Chapter 3). Here we show how the concepts of securely archived encrypted log files, exclusion/elusion, single/group authorized access can be integrated in order to provide a working framework.

Finally, the fourth part (**Chapter 6**) contains some conclusive observations and a summary of what has been done during the present research. A short summary of the improvements which could be applied on our current research is included, together with some reasoning on the ethical issues related to the field of *irrefutable administration*.

Acknowledgements

I wish to thank all the people who helped me out during these years spent at the Department of Computer Science of Turin.

Andrea Nesta, my PhD colleague and friend tragically gone in a motorbike accident two years ago. Prof. Francesco Bergadano, my supervisor, for sharing his knowledge with me and for trusting me in more than one difficult situation. Dr. Davide Cavagnino, my supervisor and friend, for his support, encouragement and kindness – a big special thank goes to him.

The whole Computer and Network Security Group: Francesco, Davide, Federica, Alessandro, Michele, Rossano, Giancarlo, Marco, Daniele, Giuseppe and all my friends and colleagues here at the Department for their kindness, honesty and cooperation.

My family, for their constant love and support. My uncle Nino and my aunt Matilde, for their example of true love which goes beyond life and death.

Samanta, for what she means to me and what – I hope ;-) – I mean to her.

Kim, for *his* constant presence, unconditioned friendship and love. Whether they have two or four feet, friends are friends.

Chapter 1
Introduction

Restricted data management is an ever-growing research and development area nowadays, mainly when it comes to standard use cases such as private documents protection for personal use, encrypted data sharing or digital signatures. It's not hard to find resources on this subject: academic literature and software houses have been dealing with the matter of data protection through encryption for some years now, but mainly for what pertains to somehow "standard" situations. Companies like PGP Corporation [PGP05] offer solutions for data encryption/sharing commonly used as of now, and several products are available for free or for commercial use. On the other hand, the circumstances get different when the use cases diverge from the "standard" aforementioned ones, and the game gets enriched by factors such as:

- Reserved data access with authorization for group of users instead of single users
- Group signature of stored data, with signer's anonymity and the possibility of addition/revocation of members from groups – but later identity escrowing
- Secure data storing with tamper-evidence service (the possibility of detecting manumissions but also addiction/removal of data)

A practical example – and a use case I will be focusing on throughout this dissertation – is that of log files (i.e. electronic archives which keep track of actions, operations and transactions taking place on a system) with the aforementioned characteristics, and with distinctive emphasis on security and privacy issues. When more entities are involved in the logging activity and a single logging facility is in use, it's clear that more issues arise: access to records must be shared, granting anonymous logging and the possibility of escrowing (identity disclosure) in case of need. Therefore, besides encryption/decryption issues we have in this case also the capabilities offered by techniques such as Secret Sharing and Group Signatures.

Secret Sharing has been studied and used in the design of the framework presented in Chapter 3. We've seen different solutions, but Shamir's proposal seemed to fit better to our context. Group Signatures have been deeply worked on. Two solutions have been devised that allow an entity to sign on behalf of a group, without revealing his identity, with the possibility of later revocation and group members addition. Chapter 4 describes those solutions, with their benefits, drawbacks and possible future improvements. From the research point of view, this thesis includes topics of relevant scientific interest. Secret Sharing and Group Signatures techniques are now a field widely explored but yet quite challenging.

The originality if this work is contained both in the pure research areas – which can be inserted in a context of never-ending growth and with yet to be explored fronts – and in the practical possibilities emerged by the proposed solutions. The field of reserved data management – e.g. log files – offers several points of interest, mainly when it comes to those real systems which could benefit from such a research.

As for applicative possibilities, several practical contexts offer the results of this research to be tested "for real", in real systems. An exemplary application resulting from the research is that of *irrefutable administration*. In many applications there are contexts in which it is necessary to check and verify the operations performed by some entity (possibly an administrator) on another entity (possibly a computer system). For example, in industrial environments some jobs are sent in outsourcing to external companies; the operations performed by the external personnel should be controlled in some way, and at the same time, the privacy of the workers must be guaranteed, with the ability to verify and link, in case of necessity, the operations performed to the person who made them. This not only goes in favor of the company, but also of the worker who is therefore more guaranteed in his tasks.

The different technologies used in this research have therefore been analyzed and opportunely mixed together in order to produce a framework which will grant the *irrefutable administration* of a remote system through a secure (and privacy-aware) logging with shared access. Of course, the framework can be extended and improved so as to grant much more than simply the administration of remote systems, in a way we will see in details in the following.

That system which has been implemented is related to the previously mentioned context, where system and network administrators, administered systems and networks are managed in order to provide a secure and reliable auditing system. The main characteristics of this system are the ability of logging all the operations that occur in a complex environment, linking these operations to the entities involved (namely, the administrator of the system and the system itself), with the guarantees that:

- the log does not immediately disclose its content, for privacy reasons (i.e. it is encrypted);
- the log's content may be examined only by the entities having the rights to perform this operation (i.e. only the authorized people, administrators or third parties, may decrypt the log entries content, with some defined modes);
- log entries cannot be directly related to the entities that are involved in the activity described in the log entry itself;
- the log cannot be modified without detection (i.e., if the log is modified this can be discovered by the auditors that will eventually check the content).

The research has therefore many fields of application, where the main problem to be solved is the logging of some activity for a subsequent control by whom is in charge of monitoring, with some important warrantees on authentication, anonymity and group/single access.

Chapter 2
State of the art

2.1 Introduction

The literature contains several technologies that can be used in order to build a secure storing of log data. This section will present the most widely known and the most efficient. Moreover, the main encryption algorithms will be presented and compared, giving a justification for the design choices made for the developed secure storing system presented as implementation to the current research in Chapter 5.

In modern systems, it is becoming more important keeping log files to be able to track down the actions taken by a system, to trace the accesses (authorized or not) to a computer or, as in our case, to keep track of all the actions made by certain users. In some cases, it is necessary to store this information in a secure and confidential manner.

The commonly used definition of log has been given by Ruffin [RUF95], which describes a log file as "a plain file where data are stored sequentially as they arrive, by appending them to the end of the file. When a problem arises in the system (e.g. a fault or an intrusion), the log is reread to find its source and/or to correct its consequences". The case we will examine uses a similar definition of log file. In fact, leaving the implementation details that force to consider the log file as a binary file containing the network dump, also in the case at hand the data is stored according to the arrival order. The usefulness of these data is to allow, later in time, the possibility to perform some analysis and investigations on them.

Ruffin makes a detailed description of logs related to the use with DBMS, but he does not examine the requirement to avoid the manipulation of the log files made by unauthorized people. Given that in modern computing systems it is often necessary to keep the log files on untrusted machines, it is important that the log file be intrinsically secure to avoid that unauthorized persons access the logs (that, in general, may contain important and confidential data).

Some approaches have been developed to authenticate or certify the content of different types of log files, possibly of large size. [BCE02] and [BCN01] present some possible methods.

Different approaches have been studied to authenticate or certify log files containing different kind of information and generally having a large size. In particular, we may cite [BCE02] which presents a technique that allows a third party to certify a large file without examining all the file content (to speed up the certification operation and/or to protect the privacy of the creator of the file), but simply examining a file sample (i.e. a small number of records).

This approach intrinsically requires that the log file already exists to proceed to authenticate it. It is also interesting to examine methodologies that are capable to keep secure every single entry of a log file while it is being created.

2.1.1 Systems using dedicated hardware

The first techniques used to prevent the modification of log file entries were not based on cryptography, but instead they used dedicated hardware. Among these methods we may recall the expensive method of printing the log on hardcopy or, equally, recording the log on WORM (Write Once Read Many) devices (also in this case having poor performance).

Obviously, these approaches are not tailored to the requirements of entities that should keep large amounts of log data for a long time. This is due to economic (costs of supports) and space (required for storing the supports) considerations, but also to system performance reasons.

For these reasons, it is necessary to find solutions that, by means of cryptographic systems, make the modification of a log file impossible without notice.

2.1.2 Systems based on cryptographic functions

A first hypothesis on the possibility to detect log file modifications has been made in [BY97] by M. Bellare and B. S. Yee: they propose to authenticate every log file entry with a Message Authentication Code (MAC).

A MAC, also known as a cryptographic checksum, is the output of a function that takes a message (in the case at hand, a log line) and a secret key as input; when verifying the authenticity of the MAC, the same function should be applied to the message and the secret key (thus the verifier must know the key used to generate the MAC). In general, MACs are computed using hash functions (HMAC is a function of this kind) or symmetric encryption algorithms (like DES-CBC).

The system by Bellare and Yee does not avoid log file manipulation, but guarantees that if an entity takes control of the system on which the logs are stored at a certain time, and tries to modify the previously logged entries, then this action will not go unnoticed. This is possible because it is foreseen that the MAC key changes over time, without leaving information about older keys. To obtain this result, the first generated key (the only one that must not be deleted and that must be kept in a safe place) is used to authenticate the log entries in a first time period of fixed duration. When the time period expires, it is generated a new key using a pseudo-random non-invertible function starting from the previous key. The previous key is then removed from the system (except the very first one).

It is easy to see that an intruder is not able to get the previous keys used to authenticate the entries. Thus, he is not able to modify the log file without notice by an authorized person (who knows the keys used to compute the MACs), because he cannot create a correct MAC for the modified entries. To avoid deletion or insertion of entries, it is necessary that the entries are sequentially numbered.

The system described is secure with respect to many known attacks (assuming the use of a robust authentication function), and also prevents the chosen message attack, given that it destroys the keys as soon as they are no more needed.

It is important to notice that if an adversary gains control of the system at time t, then from that moment there should be no confidence on the authenticity of the logs. Moreover, given that the log entries are not encrypted, the log file may be read (without detection) by any adversary.

The problem of the authentication of single log file entries by means of cryptography has been faced also by B. Schneier e J. Kelsey in [SK98] and refined in [SK99a]; they built a method to make log files unreadable for those not having the rights to access and control them, and also to detect in a fast way if the log file has been corrupted or modified.

In particular, the environment for which the method has been developed is composed by a computer U not sufficiently protected against an attack, that must keep the log files containing confidential information, and by a trusted computer T that must manage the security of the log files on U. By means of a few interactions between these two entities it is possible to prevent an adversary taking control over U at time t from reading the logs generated before t and from modifying the logs generated before t without being detected afterwards.

Also in this case, the system does not prevent an attacker from tampering with the log file, but it allows detecting whether the log file has been modified in some way. As in the paper from Bellare and Yee (not cited in the bibliography of the paper from Schneier and Kelsey), the security of the system is increased by changing over time the authentication and encryption keys; more precisely, a new key pair is created for every new log entry by applying a hash function to the previously used keys. Moreover, to avoid that the insertion or deletion of entries goes undetected, the various entries are sequentially concatenated by means of a hash chain. Consequently, it is signed the "ring of the chain" corresponding to each entry only, and not the entire entry. The security of the system is nonetheless maintained.

Schneier and Kelsey give a detailed description of the system, showing the initial creation of the log file, the proper closing of it, and how to react to sudden shutdowns of the computer keeping the log file.

Given that every entry of the log file has an associated type, the discussed papers describe a method for differentiating the access of auditors having different privileges. The auditors communicate exclusively with the trusted computer T to obtain an unencrypted entry to which they have access rights. This latter property has been studied in depth by Schneier and Kelsey in [SK99b]: in that paper they describe how to minimize the bandwidth required by an auditor to verify the authenticity of the entries, decrypt and read them.

The main problem of this approach, related to the access of auditors to the data, is that it is the trusted computer T to decide whether an auditor may access the log entries. It would be nice if the access rules could be enforced during the encryption phase instead, for example using different encryption keys for different auditors. In this way, even gaining control over T, it would be impossible for auditors to see entries, except for the ones they have authorization. On the other hand, even if T is a trusted machine, it would be useful to find methods that do not need critical nodes from the security viewpoint.

The system described by Schneier and Kelsey has been patented in [SK99c]. In that patent it is introduced the possibility (not described in detail) to use asymmetric key algorithms for the encryption. The use of asymmetric encryption was not examined in their previous papers.

One of the targets studied in this research is strictly related to the results presented in the papers from Schneier and Kelsey; in fact, the system we will discuss has some common ideas with the one just described. Nonetheless, important functionalities and characteristics will be added to increase the security and flexibility of the overall system.

In 2002 C. N. Chong, Z. Peng and P. H. Hartel analyze the study from Schneier and Kelsey, and describe a possible implementation in [CPH02]. The system they describe is based on the use of a tamper-resistant hardware (iButton) that generates and stores the secrets, avoiding to leave this critical duty to an untrusted computer (as was happening in the model described by the two previously cited authors).

Recently B. R. Waters, D. Balfanz, G. Durfee and D. K. Smetters [WBD04] made a new study on secure audit log. The problem they want to solve is to create an encrypted log that could be searched using some keywords. The approach followed to encrypt and link the log entries is very similar to the method proposed by Schneier and Kelsey; the main difference is that this method extracts some keywords from the data to be logged before they'll be encrypted. An auditor willing to search for some data has to send a keyword to a centralized trusted element; on the basis of the element's authorization policy such element may grant him the necessary information to reconstruct a key to be used for decrypting the data itself. In this paper it is not discussed how keywords could be extracted from raw data.

2.2 Hash functions and hash chains

From the just presented state of the art section on secure storing of log files, it may be seen the particular importance of hash functions and hash chains when looking for a safe method to record data. Consequently, it is useful to describe the most widely known and most efficient methods for creating hashes.

A hash function is a mapping from an arbitrary long data to a fixed length code, the so called "hash code". The implementation, be it hardware or software, shall compute the hash in a fast manner. The other constraints of a hash function are: a) to be non-invertible (i.e. one way); and b) to be collision-resistant (i.e. it is computationally hard to find a pair of values having the same hash value).

There are many algorithms that are used to implement a hash function. They have become, thanks to their characteristics, widely accepted standards. We recall here the most widely known and used today, namely MD5, SHA-1 and RIPEMD-160.

MD5 was invented by Rivest and is published in RFC1321. It produces a 128 bit hash code in 64 steps only. This algorithm has been used for years, but given the today available computers, a 128 bit hash code should not be considered sufficient. Moreover, some famous researchers of RIPE (European RACE Integrity Primitives Evaluation) have shown that it is relatively easy to find a collision for MD5.

For these reasons, those researchers developed a project in [BDP97] providing a more robust hash function, called RIPEMD-160, derived from MD5. This new function solves both the problems related to collisions and to brute-force attacks, given that the digest produced by the function is 160 bit long.

Some years before, the National Institute of Standards and Technology (NIST) publishes in [NIST95] the definition of the algorithm SHA-1. This algorithm outputs a 160 bit hash in few computation steps, being very fast. SHA-1 seems to be also collision-resistant. Recently, a 160 bit hash seems to be inadequate for some possible attacks; for this reason, the NIST has published in [NIST02] the algorithms SHA-256, SHA-384 and SHA-512, that are able to produce hashes of length 256, 384 and 512 bit respectively.

After having presented the characteristics of the most widely known hash functions, it is possible to analyze the main advantages deriving from the use of hash chains, namely the speed at which can be computed (both when generating and when verifying) and the unfeasibility to insert, delete or modify elements of the chain without this being detected with a rapid analysis.

Hash chains have been introduced for the first time by L. Lamport in [LAM81], using them for authentication systems based on one-time passwords. In particular, the idea in [LAM81] is to generate, from a known password, a chain of hashes, and to use the sequence of hashes in reverse order as a set of keys. Each key will be used for a single access. The advantage of this approach resides in the fact that not all the passwords are to be stored, but only the origin of the chain (i.e. the password known to the two parties)and a counter that keeps track of the number of accesses. The speed of computation of a hash function allows to calculate many rings of the chain without introducing large delays.

The idea from Lamport, thanks to its efficiency and effectiveness, has been applied in many contexts; besides the application in the context of secure logging, we may recall the novel system proposed by F. Bergadano, D. Cavagnino and B. Crispo in [BCC01], in which hash chains are used to authenticate a stream of data.

2.3 Symmetric key encryption

The requirement to encrypt large quantities of data has brought, in the latest years, to the development of many different cryptographic systems. Some of these systems may be useful for encrypting log files, thus in this section will be presented the main symmetric encryption algorithms, namely systems that use the same key for encrypting and decrypting data.

The main advantage of symmetric encryption systems is the speed at which can be made encryption and decryption of data. This is different from the less performing asymmetric encryption systems which will be discussed later.

There exist two categories of symmetric ciphers, namely block ciphers and stream ciphers. Block ciphers subdivide data into blocks of fixed size and then encrypt the single blocks. Stream ciphers can encrypt single bytes or even single bits. Given that block ciphers obtained a greater success with respect to stream ciphers, in the following will be discussed only the former type.

The most known symmetric encryption algorithm is DES (Data Encryption Standard), standardized by NIST in 1977, and improved until the latest definition in [NIST99]. DES is a block cipher that employs a 56 bit key. The encryption process is made up of 16 rounds, thus the key is used to generate 16 subkeys of 48 bits each, and every subkey is used in a different round. The DES encryption and decryption phases are similar; in fact the same algorithm is used for both operations, only subkeys are used in reverse order in the decryption operation.

DES has represented for many years the most secure and fastest encryption algorithm. But with the increased computing power of machines, this algorithm has become obsolete both because it uses a too small key (exposing it to brute-force attacks) and because the encryption of two equal blocks with the same key produces the same encrypted text. The latter point exposes he result to possible encrypted text analyses.

For the previous reasons, evolutions of DES have been developed. One of the most important is DES-CBC, which is based on the use of an encrypted block to modify the following block to be encrypted. This method is known in general as CBC, Cipher Block Chaining, and is widely used in many symmetric ciphers.

Another important and widely used DES evolution is Triple DES. The name itself suggests the working of the algorithm that applies three times the encryption using two or three different keys. This leads to key length of 112 and 168 bits.

At the same time DES improvements were proposed, some other algorithms were developed in order to substitute it. Among them, we may cite IDEA that uses 128 but keys and block size 64 bits.

A very interesting symmetric encryption algorithm is Blowfish, developed by B. Schneier [SCH94]. Its main features are its speed, its low memory requirements (about 5 Kbytes), the simplicity of the system (that allows easy and efficient implementations) and the possible use of different key lengths. In particular, Blowfish works on 64 bit blocks, and may use keys of length from 32 bits to 448 bits, in steps of 32 bits. This allows for a high flexibility on the degree of security one may need. Probably Blowfish is the most adaptable, efficient and fast encryption algorithm. It is also strong against brute force attacks, when using sufficiently log keys (more than 128 bits).

The last algorithm we will analyze is the one proposed by J. Daemen e V. Rijmen in [DR00] and named Rijndael. This algorithm is important because it has been chosen by NIST as the actual encryption standard [NIST01] with the name of AES (Advanced Encryption Standard). The reasons for which it has been chosen as a standard reside in its security due to the block size it uses (128 bits, and a bigger block size means a bigger difficulty in making an analysis of the encrypted text) and to the possibility of using keys of different lengths. AES allows key of length 128, 192 or 256 bits. Moreover, even if it is not as fast as Blowfish, it has good performance and efficiency.

2.4 Public key cryptography

The development of public key cryptographic systems represents one of the bigger innovations in the field of computer security. In fact these systems do not base their strength on substitution and permutation of bytes, as in symmetric ciphers. Instead, they use mathematical properties of wisely chosen functions, and require two different keys to be used, one for encrypting and one for decrypting the data. It is this latter property that lead to their name of asymmetric ciphers.

The main characteristic of these systems is that even knowing one of the two keys, it is impossible to determine the other one. This allows distinguishing the two keys, having a public key known by everyone, and a private key that must be kept secret by its owner.

This approach to cryptography gained a great success and has a widespread use, but has never substituted the symmetric cryptography systems. This is mainly due to the fact that the encryption and decryption operations with asymmetric systems are slower than the corresponding symmetric ones.

The first public key encryption system has been proposed by W. Diffie and M. Hellman, which in 1976 published in [DH76] the first mathematical formalization of a cryptographic system using asymmetric keys. This system is useful for the exchange of symmetric keys between two entities. The security of the system is based on the difficulty of the prime number factorization of large numbers and in the properties of modular arithmetic.

The first block cipher that uses asymmetric keys has been proposed in 1978 by R. Rivest, A. Shamir and L. Andelman in [RSA78], with the name RSA. Until today, this is the most widespread and used public key cryptosystem, given that it guarantees a high level of security. Also in this case the strength of the method is given by the difficulty of the factorization of a large number in primes. Moreover, the use of keys of large length (1024 or 2048 bits) makes impossible attacks based on exhaustive search.

The encryption of a message through RSA is made using the public key of the recipient; in this way, only the recipient is able to decrypt the message with his own private key.

Another interesting thing to note is the use of the RSA algorithm for digitally signing messages. In this case the signer encrypts a message digest with his private key. Anyone will then be able to verify the integrity of the message and the signer of the message, using the sender's public key.

RSA is not the only one public key cryptosystem. There is at least another approach, less known but more efficient from a computational viewpoint, named elliptic curves. The algorithms of this family have the great advantage that even using shorter keys (less than 300 bits), allowing for faster computations, they have a great degree of security. The main disadvantages are a more complex implementation and a more difficult mathematical analysis. Furthermore, the large diffusion of RSA has increased the users' degree of confidence in this algorithm, slowing the diffusion of other systems.

2.5 Secret sharing

It would be too limiting to study the techniques for making a log file inaccessible only. In fact, it is important to study also the methods that allow the auditing of the logged data.

In this case it is necessary to allow for diverse kind of accesses to different auditors having different privileges in the visualization of the information. Furthermore, it is also important to allow the access to some kind of data only to a group of cooperating auditors. According to the requirements just presented, some proposals on secret sharing will be discussed. Secret sharing means the possibility to recreate a secret only with the collaboration of many people, each one owning a part of this secret.

G. R. Blakley was the first to design in [BLA79] an abstract model useful for the sharing of a secret. His target was to find a method for keeping a copy of cryptographic keys avoiding to give many people the knowledge of the complete secret. With this aim, Blakley identifies a method based on some geometry elements. In the following a simple example will present this technique.

Let’s see an example of this method from [MOV96]: suppose we want to share a secret among n parts, requiring at least 3 to reconstruct it. Suppose also that the secret may be represented as a point into the three dimensional space. Now, build n non-parallel planes such that the intersection of any two planes defines a straight line, whilst the intersection of any three planes defines the point representing the secret. Thus, having any three out of n parts (i.e. three persons of the group put their secrets together) it is possible to discover the secret.

A generalization of this scheme is possible, thinking of m-dimensional spaces. In this case, the cooperation of m users will allow the reconstruction of the secret.

Almost at the same time as Blakley, A. Shamir presents in [SHA79] the possibility to share a secret among n individuals requiring m of them to recover the secret. His idea seems straightforward, and also effective and efficient. He presents in his short paper a still used solution to the problem, applying polynomial interpolation and some mathematical principles. The solution proposed assumes that the secret to be shared can be represented as a number. Let's call this secret D, and let's build a polynomial Q having degree m-1 and known term D.

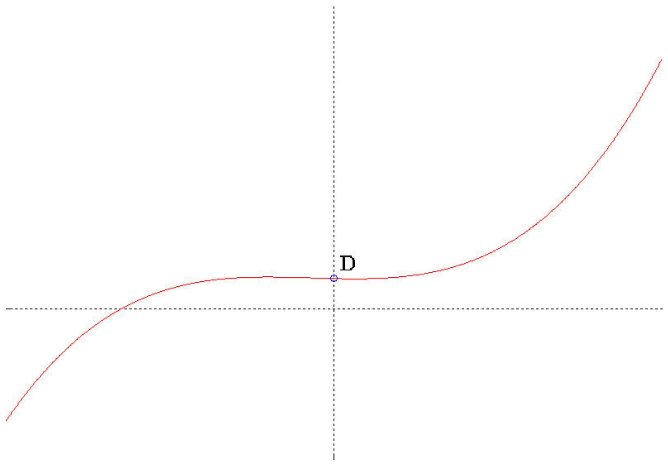

Figure 2-1: The polynomial Q

Let's call Di the evaluation of Q in i, i.e. D1=Q(1), ..., Di=Q(i), ..., Dn=Q(n).

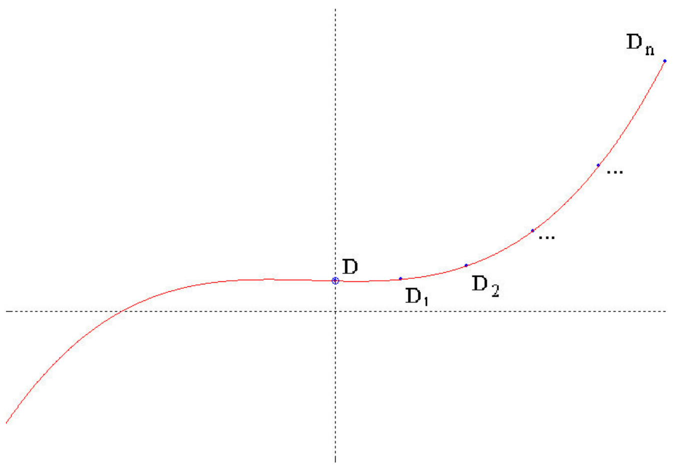

Figure 2-2: Evaluations of the polynomial Q

It may be shown that, knowing m different evaluations (not necessarily sorted in ascending order), it is possible to interpolate the unique polynomial of degree m-1 and constant term D. It may also be demonstrated that the use of less than m evaluations makes it impossible to uniquely determine the shared secret. This is a protection against small group members collusions. In the same paper, Shamir develops improvements to show a computationally efficient solution, basing the operations on modular arithmetic. The modulus to be used is a prime p greater than D and n.

The solution proposed by Shamir is intrinsically flexible to changes in the parameters n and m. Moreover, it allows to distinguish among group members, giving more priority to some members assigning them more than one different evaluation of the polynomial.

The ideas from Shamir and from Blakley have represented for many researchers a motivation to start a deeper analysis of this problem. In fact, many papers have been published on this topic, giving ideas on possible optimizations, on effective variants or solutions having a lower computational cost. Furthermore, investigations have been made on possible applications of this protocol.

2.6 Group signatures

The concept of group signatures was introduced in 1991 by Chaum and Van Heyst [CHVH91]. In that paper the authors propose four different group signature schemes. The first one provides unconditional anonymity, while the others provide only anonymity under the computational constraint. One thing to note is that it is not always possible to add new members to the group, and in some schemes the group manager needs to contact the group members to open a signature. Chaum and Van Heyst [CHVH91] introduced the notion of *group signature* that allows the members of a group to sign data on behalf of the group, in such a way that:

- only group members can sign messages;
- anyone is able to verify the validity of a group signature, but he is not able to know the identity of the signer
- in case of dispute, it is possible to "open" (with or without the cooperation of the group members) the signature to reveal the identity of the person that signed on behalf of the group;

In their paper are presented four schemes that satisfy the previously cited conditions; the proposed schemes are not all based on the same cryptographic assumptions. The same may be found analyzed in detail in [KPW96]. In some schemes it is necessary a centralized entity during the *setup* only; in other schemes every member may create autonomously his group.

Another strong evolution is represented by the work of Camenish and Michels [CS97] that presents one of the best known schemes whose strength may be shown under a strong cryptographic assumption. More work has been made by Ateniese et al. in [ACJT00] as a prosecution of [CS97] with improvements in the security and efficiency.

A big part of the proposed schemes have a signature length and/or a group public key size that depend on the group size: obviously these solutions are not adapt for large groups. Camenish has proposed some alternatives having fixed signature length and fixed group public key. These alternatives have been worked on by Ateniese et al. as previously said. Many other solutions having fixed signature size and fixed group size have been proposed, but most of them [ACJT00] revealed not provably secure (or not secure at all) or extremely inefficient.

In [ACJT00] the proposal in [CS97] was improved by making it more secure and efficient, keeping the mathematical principle on which it is based similar. With respect to [CS97], [ACJT00] uses a more efficient JOIN method for new members, and uses a registration protocol statistically zero-knowledge regarding the group member's secrets. By contrast, in fact, Camenish required the new member send to the group manager a product of his secret (a prime number with a particular form) and a random prime number; this system may be attacked, as shown by Coppersmith [CO96].

Ateniese emphasizes how the system in [ACJT00] (defined by him as belonging to "Class II") has fixed size public key and signatures. Nonetheless, that system lacks a good support to the revocation of members, and the paper discusses the issue only in the conclusions. Ateniese proposes an extension to the scheme towards a separation of works between a membership manager and a revocation manager. The revocation issue is deeply discussed in the paper [AST02], in which a CRL for the revocation of members is added to the signature scheme presented in [ACJT00]. The main disadvantages of that solution are that the CRL has a size proportional to the number of revoked members and the efficiency of the algorithm suffers for the double discrete logarithm operation needed for every signature.

Going back to the solutions proposed by Chaum, it may be observed that those schemes are inefficient due to the signature size that increases linearly with the number of group members. Moreover, adding new members requires, as most of the schemes proposed so far, a modification in the group public key. In 1997 Camenish and Stadler [CM98a] proposed a method that has a constant size for both the signatures and the group public key, using a normal signature scheme, a probabilistic encryption system semantically secure and a one-way function. With respect to the revocation, it is only mentioned the possible extension of splitting the different roles of the group manager, for example to a membership manager and a revocation manager: the first one manages the insertion of new members, the second ones deals with signature opening and revocation. In a similar way the problem is dealt with by Ateniese, Camenisch, Joye and Tsudik in [ACJT00].

Identity revocation (or group member elimination) [BRS01] is therefore a critical problem. Ateniese and Tsudik [AT99] have shown how CRLs (*Certificate Revocation Lists*) are not a good method for groups. They gave the following reasons: firstly, as group signatures are based on techniques for anonymity and *unlinkability* of signatures, a signature made (illegally) by a revoked member may be discovered only by the group manager, through signature opening, and this is not practical. Secondly, if the central authority reveals some secret information on a revoked member, to immediately notice more signature misuses, then the anonymity and the unlinkability of his previous signatures cannot be maintained. Thirdly, the decision to modify the group public key is not desirable in large groups, or in groups with frequent member turnover. Bresson and Stern [BRS01] have partially solved the problem inserting revocation information in the group signatures. The disadvantage is that in every change in the group size, the signatures increase in size (therefore not having a constant size).

As previously said, Ateniese has proposed [ACJT00] a solution to the revocation that uses a CRL. When a member performs a group signature, then he must prove not to belong to that CRL.

Another problem about revocation is that no information should be disclosed on previous signatures of revoked members. If revoked members may still be able to sign, it is necessary – to preserve anonymity and unlinkability – that no secret information on revoked members is disclosed.

What is needed by a group signature scheme is the ability to immediately revoke group members. This means that revoked members must not be able to sign on behalf of the group since the very moment in which they have been removed the group member list. In the solution proposed by Ding, Tsudik and Xu in [DTX04] the revocation of a group member is immediate; thus, he is no more able to sign after being removed from the group member list.

Another scheme based on accumulators has been proposed by Camenish and Lysyanskaya [CL02]. Their solution uses *dynamic accumulators* (that allow efficient *authorization-proofs*) together with the scheme by Ateniese et al. [ACJT00] (the latter gives efficient *ownership-proofs*). The concept of dynamic accumulator introduced in [CL02] is a variation of the accumulator proposed by Baric and Pfizmann [BP97]. The scheme allows a group member to produce a simplified authorization proof, that is, having the property that the complexity of the signature verification and group membership verification are independent from the number of currently revoked group members or total group members. The solution presented by Baric and Pfizmann in [BP97] is a generalization of the work by Benaloh and De Mare with their *one-way accumulator* presented in [BM94].

Chapter 3
Secure logging for *irrefutable administration*

This chapter presents our results on the subject of secure logging, converging in the newly born concept of *irrefutable administration*. Some possible approaches are analyzed, in order to converge into a framework which allows the logging of system administration with the constraints of privacy, authentication and anonymity mixed to some warranties about the revelation of the identity of the administrator. Let's see in details, in the following, what those concepts mean.

3.1 Introduction

In many applications there are contexts in which it is necessary to check and verify the operations performed by some entity (possibly an administrator) on another entity (possibly a computer system). For example, in industrial environments some jobs are left in outsourcing to external companies; the operations performed by the external personnel should be controlled in some way, and at the same time, the privacy of the workers must be guaranteed, with the ability to verify and link, in case of necessity, the operations performed with the person who made them.

The system proposed in the present chapter is related to the previously discussed context, considering system and network administrators, administered systems and networks, with the objective of giving a secure and reliable auditing system. The main characteristics of this system are the ability of logging all the operations that occur in a complex environment, linking these operations to the entities involved (namely, the administrator of the system and the system itself), with the guarantees that:

- the log does not immediately disclose its content (for privacy reasons), i.e. it is encrypted;

- the log's content may be examined only by the entities having the rights to perform this operation (i.e. only the authorized people, administrators or third parties, may decrypt the log entries content, with some defined modes);
- log entries cannot be directly related to the entities that are involved in the activity described in the log entry itself;
- the log cannot be modified without detection (i.e., if the log is modified this can be discovered by the auditors that will eventually check the content).

The ideas presented in this chapter have many fields of application, where the main problem to be solved is the logging of some activity for a subsequent control. The discussion is organized as follows: section 3.2 presents the terms of the problem and foresees some solutions, deeply discussed in section 3.3. Section 3.4 shows some characteristics of the solution we propose, whilst section 3.6 deals with the specific problem to allow to a set of users the access to a log line.

3.2 Preliminary considerations

In our approach we consider a log file as a journal in which information coming from various activities is stored in a set of lines; each line refers to a particular event of interest in each activity. We do not consider the physical implementation, and refer to the definition given in [RUF95].

The environment of our system is composed by administrators that perform activities on objects: these activities are logged by an entity. The job of this entity is to ensure that the content of parts of the log file is available only to the authorized people (auditors) and that this content cannot be modified without detection. We want to propose a method where there is no need for a centralized element that authorizes any new people to access a previously produced log entry. The set of people which is authorized to access stored data has to remain the same as it was when a log entry was produced. In the following we discuss techniques that can be used to obtain these objectives.

The first consideration relates to data encryption. The objective of data encryption is to allow the access to particular data only to set of users. This set may change for every logged line. Moreover, it must be possible to allow access to groups of users in which the presence of at least *n* users over *N* is required to reveal the content of a line. The chosen approach is to encrypt each line with a different key. This key is generated automatically by the system, and access to this key is given according to the kind of access we want for each user, in a exclusion/elusion strategy for the log file auditing that will be presented later.

Another goal of the system we propose is to ensure that the file cannot be altered without possible successive detection by a verifier. Thus, one objective is to avoid data forging. A solution to this requirement is to use a hash chain. A hash chain keeps the log lines linked, in the same order they were originally written, and prevents the insertion of a line between two other lines. One of the first proposals of the use of chains to connect a sequence of data is presented in [LAM81]. [BCC01] use a hash chain to link a set of data transmitted in streaming; in that paper the point that remains to be solved is how to make sure that the last element of the chain is not modified.

3.3 Possible approaches

As previously seen, for privacy reasons the data section of the log line is encrypted [BCD+04] with a randomly generated key. To record this random key for later use in auditing, we consider two possible approaches:

a. Each auditor has his own symmetric secret key: the system encrypts the random key for each auditor with his secret key;
b. Each auditor has his own pair of asymmetric public/private keys: the system encrypts the random key for each auditor with his public key. The auditor will use his private key to decrypt the random key and access the log line data;

The approach of directly encrypting the log line data with the auditor's public keys was considered, but discarded due to the computational complexity of the asymmetric encryption and the amount of data that were required to be encrypted and stored.

Let's see the structure of the log lines in the two cases a. and b. The index i runs over the log lines; k is an index that runs over the identifiers of the entities involved in the logged transaction, and j indexes the various auditors. (In this example of line structure, auditors from 0 to $j-1$ have access to the line content, auditors from j to n have not. This will become clearer throughout the rest of the chapter.)

a. $$L_i = \left\{ \begin{array}{l} TS_i , U_k , \lambda_i , \underline{E_{A_i}(D_i)} , E_{K_0}(A_i) , E_{K_1}(A_i) , \ldots , \\ E_{K_{j-1}}(A_i) , E_{K_j}(\underline{H(A_i)}) , \ldots , E_{K_n}(\underline{H(A_i)}) , \\ HC_i , S_i \end{array} \right\}$$

b. $$L_i = \left\{ \begin{array}{l} TS_i , U_k , \lambda_i , \underline{E_{A_i}(D_i)} , \alpha_{K_0^+}(A_i , R_0) , \alpha_{K_1^+}(A_i , R_1), \ldots , \\ \alpha_{K_{j-1}^+}(A_i , R_{j-1}), \alpha_{K_j^+}(R_j) , \ldots , \alpha_{K_n^+}(R_n), \\ HC_i , S_i \end{array} \right\}$$

Notice, in the preceding lines, that some parts are underlined whose particular functions will be discussed deeply. In the following the meaning of the various parts is presented:

- TS_i is the timestamp issued by a Time Stamping Authority[1] or it is a timestamp assigned by the system. If a Time Stamping Authority comes into play, then TS_i is calculated on the result of an hash function (e.g. like SHA-1 [NIST95]) applied to S_{i-1}[2] concatenated with all of the data in L_i except TS_i, HC_i and S_i. It may express the time of logging or the time of the reception of the line. Both are possible approaches. Even if the data contained in the log line already contains a timestamp, TS_i may be useful for some cross checks on the data.
- U_k is a set of data related to the log entry; in our environment it represents the identifier of the user (i.e. the administrator) that generated the data in the log line, along with an identifier of the administered system. As for the responses from the systems, this may be an identifier of the system and of the user to whom this response is sent. In order to enforce the secrecy of this field the method proposed in [WBD04] could be used.
- λ_i represents the length of data in cryptographic blocks.
- D_i are the data to be logged for the i-th line.
- A_i is the symmetric key, randomly generated, used to encrypt the data of the i-th log line.

[1] The decision about whether and how often a Time Stamping Authority has to be involved must be made according to the effectiveness of the vulnerability and threats associated with the system.

[2] This field prevents an attacker that obtains B^- at a certain point in time from being able to successfully forge any previously stored log lines.

- $K_0 ... K_n$ are the *auditor*'s secret keys, used in the approach a. that uses symmetric encryption of A_i.
- $K_0^+ ... K_n^+$ are the *auditor*'s public keys, used in the approach b. that uses asymmetric encryption of A_i.
- $R_0 ... R_n$ are random values used to preserve the elusion property we will discuss in a following section.
- $E_x(y)$ represents a symmetric encryption function that uses the key *x* to encrypt data *y*; it returns the encrypted data. A good candidate function could be AES [NIST01].
- $\alpha_{x^+}(y)$ represents an asymmetric encryption function that uses the key x^+ to encrypt data *y*; it returns the encrypted data. A function that may be used is RSA [RSA78].
- $H(x)$ is a *one-way hash* function (like SHA-1 [NIST95]).
- HC_i is the element of the hash chain for the i-th log line (see below).
- S_i is the signature of the element of the hash chain, that is, it corresponds to $Sign(B^- / HC_i)$, that is the function of digital signing HC_i with the logging system private key B^-; it returns the signature. Functions that may be used are, for example, RSA [RSA78] or DSA [NIST94].

Let's see how the element HC_i of the hash chain is computed. It is the hash of the previous log line hash (i.e. HC_{i-1}) concatenated with all the elements of the current line, except HC_i and S_i (obviously, because the first one is what we are computing, and the second one will depend on the first one). In formulas, we may write that (for both proposals):

a) $$\mathrm{HC_i} = \mathrm{H}\left\{ \begin{matrix} \mathrm{HC_{i-1}}, \mathrm{TS_i}, \mathrm{U_k}, \lambda_i, \mathrm{E_{A_i}(D_i)}, \mathrm{E_{K_0}(A_i)}, \ldots, \\ \mathrm{E_{K_{j-1}}(A_i)}, \mathrm{E_{K_j}(H(A_i))}, \ldots, \mathrm{E_{K_n}(H(A_i))} \end{matrix} \right\}$$

b) $$\mathrm{HC_i} = \mathrm{H}\left\{ \begin{matrix} \mathrm{HC_{i-1}}, \mathrm{TS_i}, \mathrm{U_k}, \lambda_i, \mathrm{E_{A_i}(D_i)}, \alpha_{K_0^+}(\mathrm{A_i}, \mathrm{R_1}), \ldots, \\ \alpha_{K_{j-1}^+}(\mathrm{A_i}, \mathrm{R_{j-1}}), \alpha_{K_j^+}(\mathrm{R_j}), \ldots, \alpha_{K_n^+}(\mathrm{R_n}) \end{matrix} \right\}$$

The first element of the hash chain, namely HC_1, is computed using as previous element a fixed and known value for HC_0 which may be recorded, without encryption, in the beginning of the log file. When a line is verified, the hash of the previous line should be trusted, thus a verification of the signature of the previous line should be performed.

3.4 Encryption and exclusion/elusion

The objective of this section is to introduce the intrinsic security of the log file when it is stored on any device. In fact, it has to be taken into account that the security of the log file should not change even if it is saved and copied for backup purposes. That is, the log file content should not be alterable (by anyone) and should not be visible by non-authorized people.

To avoid the disclosure of the content to non-authorized people we already introduced the idea of encrypting the data with a random key A_i (that changes for every line): thus, this key is used to encipher the data; afterwards, the key A_i is made available to the various auditors encrypting it with the personal key of every auditor that should have access to that data. When the key has been encrypted, then it is destroyed, and only the authorized auditors will be able to reconstruct the original data. If there are auditors that should not have access to a particular log line, then A_i is not encrypted for them; instead:

a. $H(A_i)$, a one-way hash of the key, is encrypted in case of approach a.
b. a random value R_r[3] (different for every auditor) is encrypted with the public key in case of approach b.

This is done for the following two reasons:

1. **Exclusion**: it is easy to exclude one or more auditors from accessing the log line data, simply giving them a fake key: a random number in approach b. or obtained from the right key, but through a non-invertible function, in approach a.. In the literature are presented many one-way hash functions easy to compute. The use of a different A_i for every line allows for a fine granularity in giving access to every log line only to a

[3] In some embodiments, in place of R_r the concatenation of $H(A_i)$ and a random number R_r (different for every auditor) can be used.

subset of auditors. Thus the exclusion is local to every log line. We used a one-way hash function in approach a. because we considered it as an efficient source of randomness. Due to the elusion property discussed below, we had to use a different random number for every auditor in approach b.

2. **Elusion**: it is easy to see that simply looking at the log file it is not possible to understand which auditors have access to which log lines. This is due to the fact that we encrypt the key A_i for every auditor. At this point we distinguish the two approaches a. and b. previously introduced.

    a. Access to a line depends on the possession of A_i, useful to decrypt the line, or $H(A_i)$, that does not allow access to the line. But, for the properties of symmetric encryption, it is impossible to deduce which case (if A_i or $H(A_i)$) has been encrypted for an auditor. Note that it is important to use $H(A_i)$ that changes for every line. In fact, suppose to use a constant value for those auditors that should not have access to a line. Then, encrypting a constant value using a fixed key (the secret key of the auditor) will disclose the lines that are not accessible to an auditor, simply by inspection of the log file looking for a repeated value for an auditor. Note also that the use of $H(A_i)$ has the only objective to create a random number in an efficient manner; that is, instead it could be used a random number B_i different from A_i.

    b. From the properties of asymmetric encryption it is impossible to deduce which auditor is able to decrypt the key A_i. Note the use of the random values R_r to ensure the elusion property. For those auditors having rights to access to the log line, then the key A_i is encrypted along with a random number (different for every auditor) to ensure that an auditor decrypting the key A_i is not able, through asymmetric encryption using the other's auditors public key, to deduce which of them has access to the log line. At the same time, for auditors that do not have access to a line, a random value (also in this case, different for every auditor) is encrypted with the public key of each auditor, thus the resulting value is undistinguishable from the encryption of the correct key A_i and a random value.

3.5 Group auditing

One of the objectives of the system is to give different access modes to different auditors. We have already presented a method for allowing or not the access to a line. In this section we present how to give access to a single line to a group of auditors. For example, some log lines should be decrypted only when a set of at least three auditors out of five agree on looking at its content; in this way it is possible to access the data even if not all of the auditors belonging the same group are available.

3.6 Group access to a log line

In our application, we use the method from [SHA79], with the following constraints:

- each auditor should be able to access the content of a log line both alone (if he has the rights) or with the cooperation of other auditors (if he belongs to a group of auditors that should have access to the line);
- when a group of auditors has used a secret to disclose the content of a line, then this secret must be useless if used to disclose the content of other lines; the reason lies in the fact that when a group of auditors agree in looking at the content of a line, then some of them may not agree in disclosing the content of other lines to the members of the same group;
- each auditor may belong to any number of groups (also none).

We obtain the previous results by distributing to the auditors that need a group access to a log line, a share to determine the secret A_i. That is, instead of encrypting the secret A_i for an auditor, we encrypt a part that allows the reconstruction of the complete secret Ai. This implies that to decrypt a line there may be:

- users that have access to the line as alone entities, i.e. they have $E_{K_j}(A_i)$ or $\alpha_{K_j^+}(A_i, R_j)$[4];
- users that do not have access to the line as alone entities, i.e. they have $E_{K_j}(H(A_i))$ or $\alpha_{K_j^+}(A_i, R_j)$[4];
- users that have access to the line only with the collaboration of at least *k* users, i.e. they have $E_{K_j}(\Sigma_{A_i})$ or $\alpha_{K_j^+}(\Sigma_{A_i})$, where Σ_{A_i} is the share of a secret that allows disclosing A_i with the collaboration of other *k*-1 users. Users may belong to many groups, thus having many shares of the secret (obviously, the various shares will be related to different polynomials);

Note that the three sets of users may be not disjoint (the first two are obviously disjoint). Thus, our system allows for users that may access a log line by themselves, or in collaboration with other users also, or only when other group members agree in disclosing the content of a line.

Let's see which data is saved for every auditor that potentially has access to a line:

a. $E_{K_j}\left(H(A_i), ID_{group'}, \Sigma'_{A_i}, ID_{group''}, \Sigma''_{A_i}, \ldots\right)$

b. $\alpha_{K_j^+}\left((R_j), ID_{group'}, \Sigma'_{A_i}, ID_{group''}, \Sigma''_{A_i}, \ldots\right)^4$

or, for some embodiments:

$$\alpha_{K_j^+}\left(H(A_i), R_j, ID_{group'}, \Sigma'_{A_i}, ID_{group''}, \Sigma''_{A_i}, \ldots | R_r\right)^4$$

In this example the j-th auditor has not access as individual, but only as belonging to some groups. If a user does not belong to a group (or a group does not have access to the line) then $\Sigma_.$ may be left as a set of zeroes of the right size (using a proper encryption function, all this data will preserve the elusion property).

To add an auditor to the group of auditors, it is sufficient to give him a new share based on the polynomial, encrypting this share with the auditor's key. To exclude an auditor from a group it is sufficient not to give him his share anymore.

[4] If each auditor has his own pair of asymmetric public/private keys.

To modify the minimum number of auditors necessary to disclose a log line, a different polynomial should be used, according to [SHA79].

To work properly and to be able to decrypt correctly a log line for a group, the system requires at least the following information for each group:

- a group identifier;
- the minimum number of auditors that are required to disclose the secret;
- the identifiers of all the auditors belonging to the group.

3.7 Observations on multiple groups

A question that may arise on the security of the method applied on multiple groups is: what happens if shares of different groups on the secret A_i are joined together? Do these parts allow the determination of A_i? That is, let's suppose the worst case. Imagine *m'*-1 auditors of a group (requiring *m'* auditors to compute A_i) colluding with *m"*-1 auditors of another group (requiring *m"* auditors to compute A_i). Moreover, note that the two groups may overlap.

Let's write the two polynomials we want to determine:

$$y = \alpha_{m'-1}x^{m'-1} + \alpha_{m'-2}x^{m'-2} + \ldots + \alpha_1 x + A_i$$

$$y = \beta_{m''-1}x^{m''-1} + \beta_{m''-2}x^{m''-2} + \ldots + \beta_1 x + A_i$$

The target is to determine the α values, the β values and A_i; that is, overall *m'*+*m"*-1 values. The colluding auditors have *m'*+*m"*-2 points (possibly not distinct), *m'*-1 from one polynomial, and *m"*-1 from the other polynomial. This allows to write a system of *m'*+*m"*-2 equations in *m'*+*m"*-1 variables, with the following structure:

$$\begin{cases} y_{1'} = \alpha_{m'-1}x_{1'}^{m'-1} + \alpha_{m'-2}x_{1'}^{m'-2} + \ldots + \alpha_1 x_{1'} + A_i \\ y_{2'} = \alpha_{m'-1}x_{2'}^{m'-1} + \alpha_{m'-2}x_{2'}^{m'-2} + \ldots + \alpha_1 x_{2'} + A_i \\ \ldots \\ y_{m'\text{-}1'} = \alpha_{m'-1}x_{m'\text{-}1'}^{m'-1} + \alpha_{m'-2}x_{m'\text{-}1'}^{m'-2} + \ldots + \alpha_1 x_{m'\text{-}1'} + A_i \\ y_{1''} = \beta_{m''-1}x_{1''}^{m''-1} + \beta_{m''-2}x_{1''}^{m'-2} + \ldots + \beta_1 x_{1''} + A_i \\ \ldots \\ y_{m''-1''} = \beta_{m''-1}x_{m''-1''}^{m''-1} + \beta_{m''-2}x_{m''-1''}^{m'-2} + \ldots + \beta_1 x_{m''-1''} + A_i \end{cases}$$

The target may not be reached because the system of equations is undetermined if we make the assumption that a single polynomial of degree *m* is undetermined if only *m*-1 points are available. But, to discover the shared key, it is sufficient to determine A_i: we show that this is not possible. Call the set of equations coming from the first polynomial Π and the set of equations coming from the second polynomial Θ.

Given that A_i cannot be determined from Π ([SHA79]), then reducing this set should bring to an equation of this kind:

$$c_1\alpha_j + c_2A_i = b_1$$

For the same reason, reducing Θ will lead to

$$c_3\beta_k + c_4A_i = b_2$$

where the c_m and b_n are constant values.

The system of these two equations does not allow to determine A_i because α_j and β_k are different unknown (they are coefficients from different polynomials). Thus, to answer the question we posed in the beginning of this section, even if different auditors from different groups collude to determine the shared key, they will not be able to get it unless the required number of auditors in one of the groups is reached.

The same demonstration holds also in the case in which two auditors belonging to different groups own the same share (i.e. the same point in the plane, where two distinct polynomials intersect).

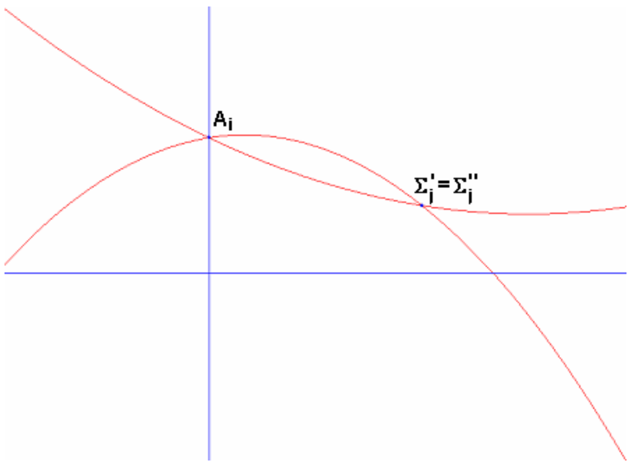

Figure 3-1: Common share of different polynomials

Chapter 4
Group signatures

This chapter illustrates the technology of “group signatures”, whose state of the art has been already presented in chapter 2.6, and some innovative solutions we’ve devised during the research we’ve carried on. Chapter 4.3 presents the first solution, based on standard signatures, putting forward in the conclusion benefits and drawbacks of the method. Chapter 4.3.8 contains a variation of the method for building group signatures mentioned above. In chapter 4.4 we describe the second solution, based on *one-way accumulators*, a newly born concept described thoroughly in this dissertation and used to build complex group signatures schemes.

4.1 Description of group signatures

Firstly, let’s precise that the term “Group Signature” (or also “Group–oriented Signatures) is not used, in this context, to denote the kind of signature in which there are groups of n participants and it’s necessary the presence of at least k out of n members to issue a valid signature. In this case, the ability to issue signatures, is granted only to a large coalition of members of a group, as introduced for the first time by Boyd under the name of *Multisignatures*. According to Desmedt [DES93], those type of signatures is nowadays usually called “*Treshold signatures*”, that is signatures where a minimal threshold of users is needed in order to issue a valid signature.

Group signature schemes are a relatively recent cryptographic concept introduced by Chaum and van Heyst [CHVH91] in 1991. In contrast to ordinary signatures they provide anonymity to the signer, i.e., a verifier can only tell that a member of some group signed. However, in exceptional cases such as a legal dispute, any group signature can be ``opened'' by a designated group manager to reveal unambiguously the identity of the signature's originator. At the same time, no one - including the group manager - can misattribute a valid group signature.

The salient features of group signatures make them attractive for many specialized applications, such as voting and bidding. They can, for example, be used in invitations to submit tenders. All companies submitting a tender form a group and each company signs its tender anonymously using the group signature. Once the preferred tender is selected, the winner can be traced while the other bidders remain anonymous. More generally, group signatures can be used to conceal organizational structures, e.g., when a company or a government agency issues a signed statement. Group signatures can also be integrated with an electronic cash system whereby several banks can securely distribute anonymous and untraceable e-cash. This offers concealing of the cash-issuing banks' identities.

A concept dual to group signature schemes is identity escrow. It can be regarded as a group-member identification scheme with revocable anonymity. A group signature scheme can be turned into an identity escrow scheme by signing a random message and then proving the knowledge of a group signature on the chosen message.

4.2 Definitions

A group signature scheme involves a group manager, a set of group members, and a set of verifiers. The group manager (for short, GM) is responsible for admitting/revoking group members, and for opening group signatures to reveal the true signers. When a potential user registers with GM, he/she becomes a group member and then can sign messages on behalf of the group. A verifier checks the validity of a group signature by using the unique group public key. We now review the definitions of group signature schemes and their security requirements as follows.

A group signature scheme is comprised of the following procedures:

- **SETUP**: on input of a security parameter, this probabilistic algorithm outputs the initial group public key and the secret key for the group manager;
- **JOIN**: An interactive protocol between the group manager and a user that results in the user becoming a new group member. The user's output is a group signing key;

- **SIGN**: A probabilistic algorithm that on input a group public key, a group signing key, and a message m outputs a group signature on m;
- **VERIFY**: An algorithm for establishing the validity of an alleged group signature of a message with respect to a group public key;
- **OPEN**: An algorithm that, given a message, a valid group signature on it, a group public key and the corresponding group manger's secret key, determines the identity of the signer;

The following properties must be satisfied by any group signatures scheme:

- **Correctness**: Signatures produced by a group member using SIGN procedure must be accepted by VERIFY procedure;
- **Unforgeability**: Only group members are able to sign messages on behalf of the group.;
- **Anonymity (or Untraceability)**: Given a valid group signature for some message, identifying the actual signer is computationally hard for everyone but the group manager;
- **Unlinkability**: Deciding whether two different valid signatures were generated by the same group member is computationally hard for everyone but the group manager;
- **Exculpability**: Even if the group manager and some of the group members collude, they cannot sign on behalf of non-involved group members;
- **Traceability**: The group manager can always open a valid group signature using OPEN procedure and then identify the actual signer;
- **Coalition-resistance**: A colluding subset of group members cannot generate a valid group signature that cannot be traced by the group manager;
- **Unforgeability of traceability**: the secrets provided by the group manager must unambiguously attest who is the real signer among two or more members that affirm to have issued a signature, even if the excluded member and the group manager are in league;

4.3 A new solution for group signatures based on standard signatures

4.3.1 Introduction

In this section we will illustrate a method for building group signatures, created in collaboration with the Network and Security Group of the Department of Computer Science of Turin. This method fits into all the required titles in terms of security, reliability and usability for the Irrefutable Administration System developed as implementation part of this thesis.

4.3.2 Description

The approach is based on standard signatures (i.e. signatures issued by a single entity, such as RSA signatures) and on a set of entities. Those entities interact in order to fulfill the properties stated in chapter [chap. Ref.]: correctness, unforgeability, anonymity, unlinkability, exculpability, traceability and coalition-resistance. Besides, thanks to the presence of an intermediate entity between the signer and the applicant, the solution here exposed owns the feature of *immediate revocation*: the revocation of a member of the group is instantaneous, unlike the traditional systems where the revoked members maintain the ability of signing even after being excluded from the group – the revocation comes out only while verifying or opening a signature.

The aforementioned properties are fulfilled considering an approach oriented to the system, more than using pure cryptography, where efficiency, security and easiness of implementation are based on the state of the art of the single components and make it easier to implement a prototype. To be more precise, although we are using an approach oriented to the system, the cryptographic side has not been left behind, as it takes a big part in the whole solution. It will be explained in detail how, with this approach, the system may adopt any public or private signature algorithm. This means that the underlying technology will, with high probability, be based on solutions which have already been analyzed and verified. The whole method will then take the correctness and principles of the underlying algorithms.

The immediacy of revocation is a definitely positive feature in a group signature context, as Ding, Tsudik and Xu illustrate in [DTX04]. In the same article they expound a solution whose model shares some basic ideas with the scheme proposed in this part of the thesis. That model, as a matter of fact, fulfills the property of *immediate revocation* by means of an intermediate entity (they call it “Mediation Server”) which filters the group signatures demands. With a single intermediate entity, however, problems connected to security and reliability of the system may arise. As a solution to this issue, also called “Single point of failure” of the intermediate entity, they introduce a secondary entity – called Group Manager – which shares with the first the role of group coordinator. A similar principle of *separation of duty* has been followed in our solution as well, as will be illustrated in the following, delegating the decision task to an entity of group managing (GME), the task of signature production to the authentication and anonymity entity (AAE) and the verification and control task to an entity of process verification (PVE).

A further difference lays, in [DTX04], into the Mediation Server, which contains also a dynamic database used for recording the signature transactions issued by the system: once a record has been recorded, it cannot be deleted. The drawback of this solution is, obviously, that the Mediation Server becomes a critical step into the system chain, because it contains all of the issued signatures. The solution I will expose in the following, on the other hand, doesn’t suffer from the same problem because the signatures themselves contain all the information necessary for their own verification and opening. Therefore, they don’t require databases or signatures archives to be put into the single entities of the system.

4.3.3 Elements of the system

In order to obtain group signature complying to the aforementioned properties, the following entities need to be introduced:

- A Group Management Entity (**GME**): it’s the entity who verifies the correct membership of the entities to the group, giving them the permission to issue signatures on behalf of the group; when a new member wants to join the group, he must first subscribe at the GME. Furthermore, when a group member wants to be removed (or must be removed) from the group, the GME attends to the removal operations;
- An Authentication and Anonymity Entity (**AAE**): it’s the entity who issues group signatures according to the group members requests; basically, it’s task is that of issuing group signatures basing on *signature requests* originating from a valid (i.e. an authorized member whose membership can be verified through methods such as public/private key authentication) group member; the peculiarity of

such a signature is that – as will be discussed in details in the following – the information is signed with a group key (owned by AAE only) but it's always possible, for the AAE, to uncover the identity of the group member who sent out the signature request. The AAE is the exclusive owner of the group key;

- A Process Verification Entity (**PVE**): it's the entity who is in charge of randomly testing – by means of random samples – the group signatures and verifying the correctness of it's contents; this entity can be seen as an AAE operations checker, it should therefore be managed by a stand-alone and separate entity; nevertheless, the owner of the PVE should shortly get rid of the pieces of information obtained during the verification process, in order to avoid the disclosure of information and hence violate the unlinkability property of the signatures (i.e. the inability of linking together two different signatures issued by the same entity);
- The group members (**M_i**): they must subscribe at the GME as group members; if the GME accepts his membership, M_i will be able to issue signature on behalf of the group, through the participation of AAE which is the real issuer of the group signature;

The AAE and the group members must possess a pair of private/public keys for this group signature scheme to be working, in a way that will be described in the following. Each public key must be put into a public key certificate, in order that the signature and therefore the real identity of the group member might be easily verified.

The correlation and interplay between the previously described entities is shown in Figure 4-1; the arrows denote the direction of the information flow, the cylinder stands for a logging device – a disk drive, for instance – where group signatures are stored in order to be retrieved in a subsequent moment.

Depending on the way in which group signatures will be used, this device could be kept always on-line and backed up at fixed time intervals.

A component not shown in Figure 4-1 is the PKI (Public Key Infrastructure), indispensable to publish the certificates containing the public keys used by the entities to make their signatures.

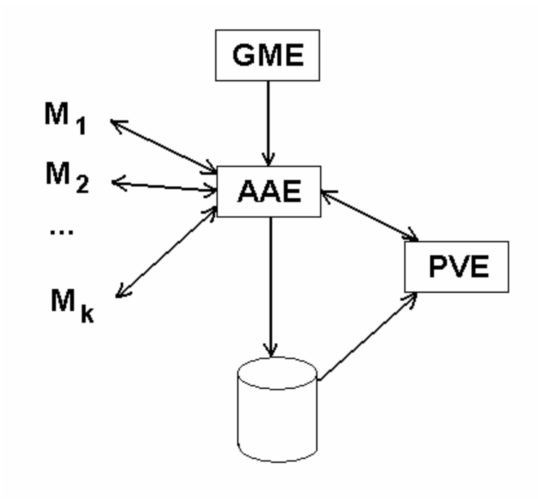

Figure 4-1: Interaction between entities

When dealing with systems where high availability is required in addition to reliability, we suggest adding some redundancy to the components required for the good operation of the system, such as in the specific context the AAE and the PVE. To make the whole process even more strong and fault tolerant it's recommended a redundancy of the GME as well. Such a redundancy becomes necessary because, in order to issue signatures, group members depend on the active real-time collaboration of the AAE, whilst the PVE should be up and running to vouch for the correctness of the whole system and of the AAE itself.

4.3.4 Communication between components

In the following we give a more detailed description of the communication taking place between the entities of the system and of the method used by the group members to issue group signatures. Firstly, the communication between the GME and the AAE must be encrypted and authenticated for those reasons:

- The encryption of the communication keeps the data exchange between the GME and the AAE private, hiding at any time the secret composition of each group (this is true for all the entities but the GME and the AAE, of course) retaining the property of unlinkability of signatures;
- The authentication of the communication guarantees the identity of the entities taking place in the data exchange and the source of the information received;

Using the channel which joins the GME to the AAE, the former gives the latter information about the group membership of each member by means of – for instance – an "ADD" message containing the public key certificate of the member to be inserted into the group.

Furthermore, the GME is entitled to remove members from groups by sending a "DELETE" message to the AAE containing the unique identification of the group member who, for a period of time stated in the message itself, will not be able to issue signatures on behalf of the group.

The communication between the different M_i and the AAE must be consequently:

- Encrypted, in order to prevent the a-posteriori linking of the signature requests arrived at the AAE to the group signatures it has produced;
- Authenticated, in order to ensure the identity of the entities involved into the communication;

For analogous reasons, the communication between the AAE and the PVE must be encrypted and authenticated.

Group signatures are written in plain text, without encryption, on the logging device, consequently the communication with this entity may be unencrypted and not authenticated because, as we will illustrate in the following, the source of the signature and the signature itself cannot be modified without positive evidence; furthermore, the group signature archived on the device doesn't provide any clues about the entity which issued it.

4.3.5 Operations of the system

In this paragraph we describe the operations the system takes charge of to issue group signatures, starting from the *signature request* advanced by a group member.

When a member M_i wants to sign a message *m* on behalf of the group, he sends a signature request for the message *m* to the AAE through the encrypted/authenticated channel, signing it with his private key.

The information sent to the AAE is therefore:

$$m, Sig_i(m)$$

$Sig_i(m)$ is the signature of the message *m* issued by M_i with his private key. In general, Sig_i may be a signature made with RSA or DSA or whichever is the preferred algorithm. The message *m* may be, as an improvement, composed line this:

$$m_o \mid t$$

In the previous suggestion, *t* is a timestamp or an increasing unique identification number, necessary to avoid replay attacks and m_o is the message itself.

When the AAE receives such message, it carries out those operations:

- It verifies how recent and "fresh" the message m is by means of the *t* component associated coupled with the message itself;
- In case of verification failure, the AAE sends a NACK (negative ACK) to M_i and closes the transaction;
- It verifies the message signature against the public key Mi gave to the PKI
- In case of verification failure, the AAE sends a NACK (negative ACK) to M_i and closes the transaction;
- In case of verification success, the AAE checks whether M_i is entitled to sign messages (i.e. it's not been revoked) against the list build by means of the ADD and DELETE messages received by GME;
- If M_i has turns out to have been revoked, the AAR sends back a negative acknowledge (NACK) and concludes the transaction;
- If M_i is entitled to issue signatures, the AAE encrypts – with a symmetrical key – the signature $Sig_i(m)$ concatenated with the group member M_i identification number (producing a data block denoted in the following as $E[Sig_i(m) \mid i])$, and signs with the group key the message concatenated with the encrypted block and a timestamp T (or alternatively an increasing identification number) used to avoid signatures replay attacks. This is the block:

$$m,\ E[Sig_i(m) \mid i],\ T,\ Sig_G(m \mid E[Sig_i(m) \mid i] \mid T)$$

The symbol | stands for concatenation of components, while the symbol *i* represents a unique identification code (unique with respect to the identification numbers connected to the certificates issued by the

CA). In a different embodiment, the identification code could as well be filled with the certificate itself.

The three elements concatenated in such way embody the group signature which the system issues on the message *m*. Everyone may easily verify this signature by means of the public key of the group (made publicly available by the PKI), and ignoring the encrypted part.

The signature may hereby be stored on permanent storage devices – for instance on hard drives as we suggested before.

In order to avoid possible coincidental correlation between a signature request advanced by a member M_i and a signature stored on disk, the AAE could, for instance, keep the signature requests arrived in the last n minutes in a memory location and, when a sufficient number of requests has been received (at least one for each entity), only then write down on disk the whole bunch of group signatures.

Particular attention must be paid to the waiting for a certain amount of time or to the necessity of messing up the signature requests, because those simple cares might be of use to avoid a correlation between the time (or the order) of the requests and a corresponding group signature. In some cases, the information like date, time, minute and second (all of them contained in T) could be removed from the group signature for safety, considering that the system might be liable to replay attacks, unless a unique increasing identification number is used for each signed object.

In case of dispute, the AAE may "open" the signature and thus give evidence of the group member identity who issued the signature. The opening request should come from an authorize entity, to prevent attacks and private information disclosure. The opening of the signature is carried on by the AAE according to those steps:

- He verifies the external signature issued by himself.
- He decrypts (with a secret symmetrical key) the encrypted part contained in the signature

The second item allows the AAE to retrieve the identification code associated to the signer of the message *m* and his original signature. This is necessary to unfold the real identity of the signer.

Notice that any modification to the group signature subsequent to it's creation (while, for instance, it's stored on a storage device) immediately invalidates the signature, in a way similar to what happens to standard signatures.

This latter observation must be kept in mind when considering the responsibility of the AAE entity, which is limited to the correct execution of the task above mentioned of creating the group signature. It's of course impossible for anyone to produce valid group signatures without the intermediation of the AAE.

4.3.6 Validation/verification of system operations

The system here exposed contains a stand-alone entity with the sole duty of verifying by means of random samples the signature and consequently the correct efficiency of the AAE. The PVE (Process Verification Entity) checks the consistency of the signatures asking the AAE to open a small and randomly chosen number of group signatures issued in different times. Put in practice, the PVE obtains with a random decision a signature issued by the AAE, verifies the external signature (Sig_G) and, in case of correctness, asks the AAE (through the encrypted communication channel) to open such signature. The opening of the signature gives the PVE the unencrypted signature Sig_i of the group member, signature which is immediately verified. If the signature turns out to be correct, the AAE is regarded as safe and reliable, otherwise it's regarded as broken and the whole system must be immediately stopped. If Sig_i or Sig_G are revealed as corrupt, in fact, that could mean that the group signature is not valid (and the AAE could therefore be corrupted).

Just after the correctness verification of a signature, the PVE must destroy the information concerning the open signature, so as to avoid any possibility of linking between signatures.

Notice that, because of the criticality of the operations executed by the PVE, all the data exchanged between the PVE and the AAE must be encrypted and authenticated.

4.3.7 Benefits and drawbacks

In this section we will discuss the system described in the previous paragraphs, taking into account benefits and drawbacks. In the following there is a list of the benefits of the above described solution:

- It is possible to add or remove members to/from the group in real time: the GME is in direct communication with the AAE and is entitled to send to the latter the information about the group structure;

- It is possible to verify, by means of random samples, the issued signatures, so as to detect eventual manumissions;
- The issued group signature requires a small amount of space where will be stored the original, encrypted signature of the group member. Consequently, the total size of the resulting group signature depends only on the size of the standard signatures used for signing;
- Concerning the previous item, the signature of the above exposed schema requires a small amount of space, mainly when compared to some other group signatures schemes; furthermore, the system may adopt any signature algorithm whose properties have already been – for instance – analyzed and verified. There is no need, therefore, of a new implementation of signature algorithm and the consequent robustness test;
- By means of the group public key only and of the data stored on the repository (which may be for instance a disk drive) it's possible to verify the group signature issued on a message, without taking into account the encrypted part of the signature which will be indispensable for the opening of the signature. Furthermore, the signature contains also, encrypted, the information necessary to reveal the identity (also referred to as "identity escrowing") of the group member who has signed the message (for the decryption process the collaboration with the AAE is mandatory). Notice that the encrypted part contains the original first signature of the group member, and hence his identity as well.

The main drawback of this method is the necessity of an intermediate entity, the AAE, entitled to sign on behalf of the group. As discussed before, this entity is involved in the process for each and every signature. As a consequence, the AAE is prone to become a kind of bottleneck during the operations of the system but, on the other hand, it must be clear that there is the possibility of implementing such entity on an independent device, where the whole computational strength can be devoted to the signature process.

This trick makes up for the before mentioned drawback and helps keeping secret the key used for the encryption and for the signatures in a secure and protected environment (for instance in a *tamper-evident* device).

Further remark about the suggested method is that the computational power of the group signature algorithm is directly proportional to the complexity of the algorithm. More precisely, a group signature thus requires two standard signature, one encryption and one signature verification process.

4.3.8 Variant of the first solution

As described in the previous paragraph, the solution is based on an intermediate entity (the AAE) with the constraint of being always on-line and reachable, because the signature process requires its direct intervention.

In the situation where it's more desirable to release the constraint of the AAE availability, in exchange of less guaranties about the *unlinkability* property above described, a variant of the solution may be adopted. That variant has been designed in order to decrease the necessity of a repeated contact with external entities for each and every signature, the whole operation results therefore more independent.

4.3.8.1 Pseudonym certificates

The modifications to the above described solution consist in joining together the AAE and the GME encompassing them in a sort of "Pseudonym Certificates Emitter" (from now on referred to as PCE) with the charge of giving the group members entitled to sign on behalf of the group some certificates. There will be no more AAE signing messages on behalf of a group member A_i, but the member itself will sign by means of the pseudonym certificate (called for briefness PsC) received from the PCE. In details, the group member will sign using the private key connected to the public key certificated by the PsC.

For the initial request, the group member will testify his identity by means of a personal certificate (here called CRT) released by a CA authorized and acknowledged by the PCE. This latter will provide the group member with a pseudonym certificate only in the case in which there is no revocation upon the partnership and if the certificate turns out to be valid. This means that, in practice, the group member has been recognized by the system and he is entitled to sign on behalf of the group.

The relevant feature of this variant is that the PsC received from the PCE can be used by the user A_i for the signature of *more than one* message. This becomes a point of strength because it solves the problem of unreachability of the PCE and makes more easy the signature process. The more self-evident drawback is that, in case of re-use of the PsC received for more than just one signature, the property of *unlinkability* of the issued signatures is not kept true. However, this is a parameter which can be adjusted to the requirement, as it is always possible to ask for a new

PsC whenever the need arises. Of course, the duration of the PsC emitted by the PCE must not be too extended in time or number of allowed signatures, in order to permit an easy and immediate revocation of the group members as will be described in the following.

Let's summarize, in general, the steps which are necessary for the signature of one or more messages by a member A_i:

1. The member A_i generates a pair of keys (N^+, N^-), which will be used to sign the messages on behalf of the group;
2. The member A_i asks the PCE for a PsC, providing together with the request)R) the public key created during step 1 (N^+). The request is signed by A_i with his private key K^- (not to be confused with the key generated during step 1) and attaching the public key certificate acknowledged by the CA (CRT). Attached is sent also the signature made with the private key produced at step 1 of the signature made by A_i with his own private key K^-;

PsC-Request = $CRT, R(N^+), Sig_{K^-}(R(N^+)), Sig_{N^-}(Sig_{K^-}(R(N^+)))$

3. The PCE provides the member A_i with a pseudonym certificate PsC containing the public key generated during step 1 (N^+), possibly a timestamp (T), and some encrypted data. Those encrypted data must contain the identity of A_i, the request and pertinent signatures and certificate received by A_i during step 2 (PsC-Request). The symmetrical key (PCES) used for the encryption of the above listed data must be exclusive property of the PCE and generated on that purpose and different for each PsC issued. That symmetrical key must then be encrypted with a public key $PCEK^+$ whose corresponding private key is kept secret by the entity which is entitled to open the signatures. Immediately after issuing the PsC, both the PCES and the PsC-Request ought to be removed from the system PCE. The PsC may then be made public or if needed also attached to the signature itself. That cerfificate will in fact be necessary for the signature verification process. In the following are listed the fields which make up the PsC certificate:

$$PublicKey = N^{+}$$
$$Extension\_1 = E_{PCES}[PsC - \mathrm{Re}quest]$$
$$Extension\_2 = E_{PCEK^{+}}[PCES]$$
$$Extension\_3 = Version, Extension\_1Length, Extension\_2Length$$

4. The group member A_i will then be able to issue his signature on messages using the private key generated during step 1;
5. The verification of validity of the signature is made using the PsC certificate produced at step 3;
6. The opening of the signature involves the decryption of the symmetrical key contained in the PsC with the private key of the authority which is entitled to open signatures. The symmetrical key hereby obtained will be used to decode the encrypted data contained in the pseudonym certificate PsC coupled with the signed messages;

The schematic representation of the solution variant is sketched in Figure 4-2.

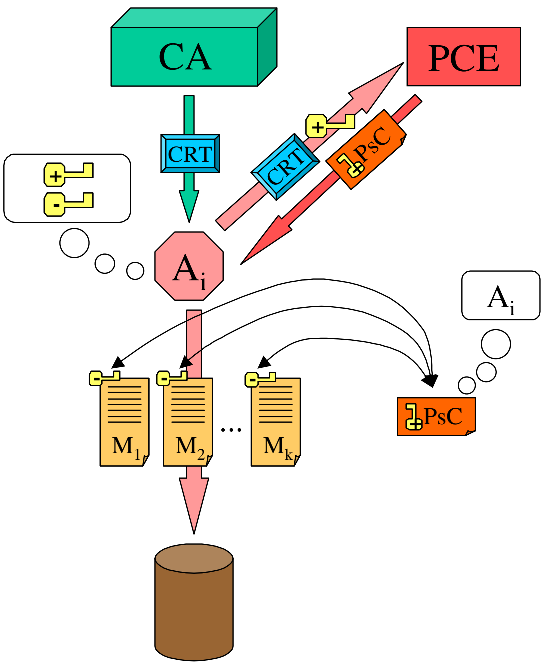

Figure 4-2: Schema of the interaction between components

As pointed out by step 6, the pseudonym certificate is sufficient to reveal the real identity of the signer, because it contains in encrypted form the original request issued by the group member. One immediate benefit is that now the AAE no longer exists, in favor of what is now called "PCE", an entity with lower responsibility because the signature is in this variant issued by the member itself.

The self-evident drawback is in this case the possibility, for a group member, to keep on signing messages even after being revoked – because with this variant the signatures do not require direct intermediation of any entity. Of course, although they are practically doable, the signatures issued with revoked certificates will not pass the verification process later on. The idea is, indeed, that of emitting certificates with short validity duration, so as to let the verification be made basing on the duration of the certificate, containing by itself a time value issued by the PCE and verified by some central server. Setting, for instance, a one day duration, it's possible to revoke group members with effect starting from the day following the revocation.

4.3.8.2 Properties

We'll see now how the variation of the solution presented in the previous chapters complies with the properties of the group signatures, failing only in some cases when it comes to the property of *unlinkability*:

- **Correctness**: signatures generated by group members are valid and verifiable by means of the PsC issued by the PCE. The PsC is obtained in exchange following the request – signed with his own certificate acknowledged by the CA – issued by the group member;
- **Unforgeability**: only the group members are entitled to produce valid signatures for messages on behalf of the group. Revoked members can still issue signatures, but they are not verifiable after the time slice of validity of the PsC assigned;
- **Anonymity** (or **Untraceability**): given a group signature, it's under computational constraint infeasible for anyone but for the PCE retrieving the real identity of the signer. Such information is, in fact, stored in the PsC associated with the signature, encrypted with a symmetrical key known only to the PCE;
- **Exculpability**: neither a coalition of members nor the group manager itself may be able to sign on behalf of other members of the group. The PCE, in fact, issues a PsC only after receiving a valid certificate request

authenticated through a Certification Authority known both to the signer and to the PCE. Furthermore, the PsC contains the real request sent (and signed with his own private key acknowledged by the CA) by the group member to the PCE, encrypted symmetrically with a password known to the PCE only. As a consequence, even if the PCE happened to issue corrupted certificates, it would not be able to put into those certificates the fake member requests – he doesn't possess their private keys as a matter of fact;

- **Traceability**: the group manager is in all cases able to open valid group signatures and to reveal the real identity of the signer (identity escrowing). The PsC contains, indeed, the identity of the signer paired with the original request he issued to the PCE in order to obtain the PsC (plus some data such as for instance the period of validity of the certificate). Notice that those data are encrypted with a symmetrical key – or the private asymmetrical key in case of different embodiment – known only to the PCE;
- **Unforgeability of traceability**: the secrets involved in the signature and in a part managed by the group manager is sufficient to prove without ambiguity who is the real signer of a signature whose property is asserted by two members. This must be true even if the excluded member and the group manager were found to be allied;
- **Coalition-resistance**: no subset of group members is able, joining and mixing together their secrets, to produce a valid group signature that the group manager is not capable of opening. The signature has, to be considered valid, to be associated to a valid PsC, issued by the PCE and with timestamping coherent with the validity period. It's not therefore possible for any member to forge a PsC because he doesn't possess the private key of the PCE with which he should sign the certificate;

We conclude with the *unlinkability* property, which results in this case dependent on the signature policy and PsC assignment adopted.

- **Unlinkability**: given two group signatures, it's under computational constraint infeasible for anyone but for the PCE telling whether the two signatures have been issued by the same signer *only if* the signature policy adopted is that of requiring a different PsC for each and every signature. In case of re-use of the same PsC, the property is not kept, because getting the PsC means being able to tell whether two signatures have been issued by the same entity

4.3.8.3 Comparison of the variant to the original method

In this section the variant to the system is analyzed under the benefits/drawbacks point of view showing how, roughly speaking, this variant comes out to be quite similar to the original version but for the tradeoff between unlinkability and high availability of the signing process.

- As in the original schema, it's still possible adding or revoking member to/from the group in real-time: the PCE assigns the pseudonym certificates only to those who are entitled to sign on behalf of the group;
- Unlike the original schema, the group signature issued doesn't require additional slots to keep trace of the original, encrypted, group member's signature. It's the group member itself who issues the real signature on the document, not an intermediate entity on behalf of him;
- The signature issued in the latter variant takes considerably less space (mainly when compared to other group signature schemes); furthermore, the solution here exposed may adopt any signature algorithm without complications, with the advantage that such algorithms should be chosen among those which have been proved and tested to be robust and efficient. As in the previous solution, there is no need of implementing a brand new signature algorithm, and to perform any kind of robustness test;
- By means of the group public key only and of the Pseudonym Certificate (in addition to the CA certificate, of course) it's possible to verify the group signature issued on a message, without taking into account the encrypted part of the signature which will be indispensable for the opening of the signature. Furthermore, the PsC contains also, encrypted, the information necessary to reveal the identity (also referred to as "identity escrowing") of the group member who has signed the message (for the decryption process the collaboration with the PCE is mandatory). Notice that the encrypted part contains the original first signature of the group member, and hence his identity as well.

The main drawback of the original solution is the need of an intermediate entity, the AAE, with the task of signing on behalf of the group (and of the member who required the signature, of course). We showed how the presence of this entity could turn out to be – if no additional care was taken – a sort of bottleneck or single failure point for the whole system. The variant above described can overcome the unreachability of the intermediate entity allowing the member to sign more than one document with the same private key, and therefore the same pseudonym certificate. It's consequently possible to implement a policy which forces the member to require a new PsC each *n* signatures, or for instance once every beginning of day. What's more, the policy could state that members should ask for new PsC for *each* signature *but* when there are connectivity problems with the PCE. This latter example is worthy of being chosen first, because the re-use of the same key and of the same certificate invalidates the property of *unlinkability* of signatures: the certificate in fact may allow an entity to tell which signatures have been issued by the same member (with the same certificate). In conclusion, it's plain clear how the two solution are based on a tradeoff between *unlinkability* and availability of the signature process.

Further consideration, similar to the original approach, is that also in this case the computational complexity of the algorithm is directly proportional to that of the standard signature algorithm chosen for the "low-level" signatures.

4.4 A new solution for group signatures based on one-way accumulators

4.4.1 Introduction

In this section we will present a second method (third actually, as the variant of the previous method can be considered as a second version) for building group signatures, created in collaboration with the Network and Security Group of the Department of Computer Science of Turin. This method is not based on standard signatures but, as we will see in the following, evolves a concept quite new in the field of computer research called *one-way accumulators*.

4.4.2 Description

The group signature scheme we propose [BCD+05] in this second solution is based on the concept of one-way accumulators, as presented in [BM94] (in the last section, this paper briefly comments on an "effective method of forming collective signatures"). A function g that is a one-way accumulator produces a value w computed by the application of g which is independent from the order of the y_j values (note also the starting x).

$$w = g(g(g(...g(g(g(x, y_1), y_2, y_3,...), y_{t-2}), y_{t-1}), y_t)$$

As function g [BM94] suggests to use

$$g_n(x, y) = x^y \bmod n$$

where n is a large rigid integer $(n = (2p'+1)(2q'+1)$, with $2p'+1$ and $2q'+1$ primes, p' and q' odd primes, $|2p'+1| = |2q'+1|$).

In our application we propose to use a modified function for g, to avoid possible attacks due to the use we do of this function (see observation I in chapter 4.4.5.1). The modified function is:

$$f_n(x, y) = x^{b(y)} \bmod n$$

where *b* is an appropriately chosen hash function (as we will see later, the result of this hash function should be an odd number; thus, *b(y)* may be equal to *h(y)* OR *1*, where *h(y)* is a standard hash function):

$$b(y) = h(y) \text{ OR } 1$$

4.4.3 Proposed Group Signature Scheme

The generic *i*-th member of *m* group members is A_i. GM is the group manager. Let's see in the following the description of the steps of the process.

4.4.3.1 System bootstrap

The system bootstrap basically consists of the following steps:

1. A_i generates a set of N *asymmetric key pairs* (e.g., for One Time Signatures, RSA, or DSA).
2. A_i sends (through an encrypted channel) to GM the set of public keys $K_{i,1}$, $K_{i,2}$, ..., $K_{i,N}$. These keys are to be used to issue the group signatures. Each key is signed with the member's secret key S_i and with the corresponding secret key[5] $K^S_{i,j}$ (the secret key corresponding to the public key $K_{i,j}$):

   $$AE_{i,j} = Sig(S_i, K_{i,j}), Sig(K^S_{i,j}, K_{i,j}) \quad for\, 1 \le j \le N$$

   where *Sig(x,y)* is the signature of y using private key *x*.[6]

3. GM collects all the public keys from all members, verifying all the signatures using the members' public keys P_i's.
4. GM generates the One-Way Accumulator [BM94] of the public keys, using a secret *X* (*X* is considered 'mod *n*'). *GK* is the group public key, then it is signed by GM and published along with the modulo *n*.

   $$GK = f_n(X, K_{1,1}, K_{1,2}, ..., K_{1,N}, ...\ K_{i,1}, K_{i,2}, ..., K_{i,N}, ..., K_{m,1}, K_{m,2}, ..., K_{m,N})$$

---

[5] The certificate of A_i's public key P_i is made available through a PKI.

[6] This is a bit different from Observation II in chapter 4.4.5.3, where it was suggested to use the private key associated with $K_{i,j}$ to sign $K_{i,j}$.

5. GM sends (encrypted and authenticated) to every group member a partial accumulator, using all of the other m-1 group member's public keys. A_i receives:

$$C_i = f_n(X, K_{1,1}, ..., K_{1,N}, ... K_{i-1,1}, ..., K_{i-1,N}, K_{i+1,1}, ..., K_{i+1,N}, ..., K_{m,1}, ..., K_{m,N})$$

6. Moreover, GM sends (encrypted and authenticated) to every member A_i:

$$Enc_{GM}(AE_{i,j}), Sig_{GM}(K_{i,j}, Enc_{GM}(AE_{i,j})) \quad \text{for } 1 \leq j \leq N$$

where $Enc(x)$ is the symmetric encryption of x.

4.4.3.2 Signing by A_i (SIGN)

A_i uses one of the private keys $K^S_{i,j}$ to sign a message M (to enforce the unlinkability property, A_i uses a key that was never used before), producing $Sig(M)$. A_i computes the One-Way Accumulator of C_i along with its public keys, except for the public key $K_{i,j}$, associated with $K^S_{i,j}$.

A_i publishes (without any contact with GM):

- M, $Sig(K_{i,j}, M)$ [i.e. the message M and the signature of the message with the public key $K_{i,j}$]
- $K_{i,j}$, $PGK_{i,j} = f_n(C_i, K_{i,1}, ..., K_{i,j-1}, K_{i,j+1}, ..., K_{i,N})$ [i.e. the public key $K_{i,j}$ and the one-way accumulator of C_i and all of the public keys but $K_{i,j}$, the one used to issue the signature]
- $Enc_{GM}(AE_{i,j})$, $Sig_{GM}(K_{i,j}, Enc_{GM}(AE_{i,j}))$ [i.e. the token related to $K_{i,j}$ received together with the others by the GM at step 6 of the previous paragraph]

4.4.3.3 Verification by anyone (VERIFY)

It is possible to verify the signature produced by any group member by checking:

- that $Sig(K_{i,j}, M)$ is the signature of M using $K_{i,j}$
- that $GK = f_n(PGK_{i,j}, K_{i,j})$
- that $Sig_{GM}(K_{i,j}, Enc_{GM}(AE_{i,j}))$ is valid.

4.4.3.4 Identification of signer (OPEN)

GM verifies the signature (as would have done anyone, see previous section), then decrypts $Enc_{GM}(AE_{i,j})$, and using the identifier field of the signature identifies the signer. Using $AE_{i,j}$ GM may prove to a third party that $AE_{i,j}$ contains the signature of $K_{i,j}$ made by A_i, and that A_i possesses the secret key associated with $K_{i,j}$.

4.4.3.5 Member addition

When a new member A_z wants to be added to the group, it produces W asymmetric key pairs (e.g., for One Time Signatures [REY02] [PER01], RSA, or DSA), and sends the necessary data to GM, as has every other member did at system bootstrap.

GM prepares the C_z for the new member A_z:

$$C_z = f_n(\boldsymbol{Y}, K_{1,1}, K_{1,2}, ..., K_{1,N}, ... K_{i,1}, K_{i,2}, ..., K_{i,N}, ..., K_{m,1}, K_{m,2}, ..., K_{m,N})$$

C_z is computed with all the other member keys and a starting value Y obtained as follows:

$$Y' = \mathrm{rad}(b(K_{z,1}), \boldsymbol{X}) \bmod n$$
$$Y'' = \mathrm{rad}(b(K_{z,2}), Y') \bmod n$$
$$Y''' = \mathrm{rad}(b(K_{z,3}), Y'') \bmod n$$
$$......$$
$$\boldsymbol{Y} = Y^{(W)} = \mathrm{rad}(b(K_{z,W}), Y^{(W-1)}) \bmod n$$

$\boldsymbol{Y}$ is the root modulo n of $\boldsymbol{X}$ computed using all the (hashed) new keys $K_{z,1}$, $K_{z,2}$, ..., $K_{z,W}$, that is:

$$\boldsymbol{X} = f_n(\boldsymbol{Y}, K_{z,1}, K_{z,2}, ..., K_{z,W})$$

The meaning of the function rad() is the following:

$$u = \mathrm{rad}(s, t) \bmod n \qquad \text{implies that} \qquad t \equiv u^s \pmod{n}$$

Computing rad() is feasible only if knowing the factorization of n. This is known as the *RSA problem*, as we can find described, for example, in [MOV96] § 3.3: "Given a positive integer n product of two distinct odd primes p and q, an integer c, and a positive integer e such that gcd$(e, (p-1)(q-1)) = 1$, find m integer such that $m^e \equiv c$ (mod n).

As shown in [MOV96] § 8.2.2, this problem can be easily solved if the factoring of n is known. GM knows the factoring of n. In our case, to compute the e-th root we need that gcd$(e, (2p'+1-1)(2q'+1-1))$ = gcd$(e, 4p'q') = 1$, where $e = b(y)$. It is then possible to determine the inverse d of e modulo $\phi(n)$, and compute $m = c^d$ mod n, being m the e-th root of c. To reduce the probability of having a gcd different from 1, then the result of the hash function is OR-ed with 1, giving an odd number. If the gcd is different from 1 (and equal to p' or q'), then the key used to compute the hash should be discarded. This event should be very improbable, if p' and q' are large primes, or impossible if, for example p' and q' are 512 bit long and the result of the hash function is 256 bit long.

Given that the group public key GK is now the one-way accumulator of C_z together with all of A_z's public keys $K_{z,1}, K_{z,2}, ..., K_{z,W}$:

$$\begin{aligned} GK &= f_n(\mathbf{X}, K_{1,1}, K_{1,2}, ..., K_{1,N}, ...\ K_{i,1}, K_{i,2}, ..., K_{i,N}, ..., K_{m,1}, K_{m,2}, ..., K_{m,N}) = \\ &= f_n(\mathbf{Y}, K_{z,1}, K_{z,2}, ..., K_{z,W}, K_{1,1}, K_{1,2}, ..., K_{1,N}, ...\ K_{i,1}, K_{i,2}, ..., \\ &\quad K_{i,N}, ..., K_{m,1}, K_{m,2}, ..., K_{m,N}) = \\ &= f_n(\mathbf{Y}, K_{1,1}, K_{1,2}, ..., K_{1,N}, ...\ K_{i,1}, K_{i,2}, ..., K_{i,N}, ..., \\ &\quad K_{m,1}, K_{m,2}, ..., K_{m,N}, K_{z,1}, K_{z,2}, ..., K_{z,W}) = \\ &= f_n(C_z, K_{z,1}, K_{z,2}, ..., K_{z,W}) \end{aligned}$$

then

$$C_z = \text{rad}(b(K_{z,W}),..., \text{rad}(b(K_{z,2}), \text{rad}(b(K_{z,1}),GK) \bmod n) \bmod n) ...) \bmod n \quad (*)$$

4.4.3.6 A possible improvement: incremental group creation

These considerations suggest a ***method for creating the group*** which might be used instead of the bootstrap previously described in chapter 4.4.3.1. The group manager chooses a public key GK (mod n), and publishes it as previously detailed. Every time a member asks to be added to the group, then that member is requested to send the keys to GM, that replies with the same information, but computes the value C_z as in $^{(*)}$. This mode of operation implies that GM **does not need** to store the public keys of the group members.

The value of C_i for any other group member A_i obviously does not change, as well as the group public key GK.

4.4.3.7 Adding more keys for a group member

The addition of more keys to a group member is dealt with as adding a new group member, so for this subject please refer to paragraph 4.4.3.5.

4.4.3.8 Properties

The proposed group signature scheme owns the following properties:

a. Signing does not require any contact with GM.
b. The group public key GK has a fixed size.
c. The signature has a fixed size.
d. Private keys do not change when new members are added.
e. Private keys do not change when more keys are given to members.
f. The group public key GK does not change when new members are added.
g. The group public key GK does not change when more keys are given to members.
h. The group members need not change their private keys when new members are added.
i. The group members need not change their private keys when more keys are given to a member.

4.4.4 Adding revocation to the current solution

The revocation feature can be added to the current solution by slightly modifying the functions and the principles used in it. In this paragraph, we will discuss our suggestion on how the revocation can be realized and what differences are introduced with respect to the current solution. Some properties must be listed first:

Property 1

Given $\gcd(a, n) = 1$, (i.e. a and n relatively prime),

$$\text{if } z = t \bmod \phi(n), \text{ then } a^z = a^t \bmod n.$$

Obviously property 1 can be applied iteratively, that is, having a^z,

$$(a^z)^w = a^{zw},\ \gcd(a, n) = 1$$
implies that
$$\text{if } s = (zw) \bmod \phi(n) \text{ then } a^s = (a^z)^w \bmod n$$

Property 2

$$(d \bmod n)(e \bmod n) = (de) \bmod n$$

Property 3

$$\begin{gathered}(a^b \bmod n)^c \bmod n = \\ ((a \bmod n)\ (a \bmod n)\ ...\ (a \bmod n))^c \bmod n \\ = ((a \bmod n)^b)^c \bmod n = \\ = (a \bmod n)^{bc} \bmod n = \\ (a^{bc} \bmod n) \bmod n = \\ a^{bc} \bmod n\end{gathered}$$

Property 4

Computing roots modulo n is related to the *RSA problem* (see, for example, [MOV96] § 3.3 or chapter 4.4.3.5). As mentioned before, it may be shown that this problem is easily solved if the factors of n are known, and GM knows them. We will now see how the functions previously described can be modified to support revocation.

4.4.4.1 System bootstrap

GM computes the large rigid integer n ($n = pq = (2p'+1)(2q'+1)$, with $p = 2p'+1$ and $q = 2q'+1$ primes, p' and q' odd primes, $|2p'+1| = |2q'+1|$). We may choose $|p| = |q| = 512$ *bit*.

Let's define the notation partially used in the previous paragraphs as well:

- **h** : hash function, h: $\{0,1\}^k \rightarrow \{0,1\}^{256}$
- **b** : b(x) = h(x) OR 1 (see later)
- **Sig** : Sig(a, e) represents the signature of e made using the private key a.
- **Enc** : $\text{Enc}_A(x)$ is the encryption of value x using a secret key known to A.

- **R** : R(a, e) mod n represents the a-th root of e computed modulo n, i.e. e = R(a, b)^a mod n.
- ***$f_n(x, y)$*** = $x^{b(y)}$ *mod n* : one-way accumulator as defined in [BM94], with slight modification, for the application at hand;

$$f_n(f_n(x, y_1), y_2) = (x^{b(y_1)} \bmod n)^{b(y_2)} \bmod n$$

and sometimes we will write

$$f_n(f_n(x, y_1), y_2) \quad as \quad f_n(x, y_1, y_2)$$

From property 3:

$$f_n(f_n(x, y_1), y_2) = x^{b(y_1)b(y_2)} \bmod n$$

Moreover, from property 1, if gcd(x, n) = 1 then it is possible to find $\beta = (b(y_1)b(y_2)) \bmod \phi(n)$ such that

$$f_n(f_n(x, y_1), y_2) = x^{b(y_1)b(y_2)} \bmod n = x^{\beta}$$

To create the group, GM must choose a random GK (mod n) such that $\gcd(GK, n) = 1$ (the reason for this restriction will be clear later) and publish it signed, along with an empty CRL (Certificate Revocation List).

4.4.4.2 Adding a member to the group

When a participant A_i wants to join the group, A_i generates a set of N *asymmetric key pairs* that will be used in the various signatures.

A_i encrypts and sends to GM the set of N public keys $K_{i,1}$, $K_{i,2}$, ..., $K_{i,N}$, each one signed with A_i's secret key S_i (with certificate of A_i's public key P_i) and also signed with the corresponding secret key:

$$AE_{i,j} = Sig(S_i, K_{i,j}), Sig(K^S_{i,j}, K_{i,j}) \quad for\, 1 \le j \le N$$

where *Sig(x,y)* is the signature of y using private key *x*

At the reception of the N public keys, GM computes:

$$Y' = R(b(K_{i,1}), \mathbf{GK}) \bmod n$$
$$Y'' = R(b(K_{i,2}), Y') \bmod n$$
$$Y^{(3)} = R(b(K_{i,3}), Y'') \bmod n$$
$$\ldots\ldots$$
$$\boldsymbol{C_i} = Y^{(N)} = R(b(K_{i,N}), Y^{(N-1)}) \bmod n$$

It is easy to see that:

$$GK = f_n(C_i, K_{i,1}, \ldots, K_{i,N})$$

The construction of $b()$ ensures that gcd$(b(), n) = 1$, hence the existence of the roots. GM sends to A_i the value C_i through an encrypted and authenticated channel along with

$$Enc_{GM}(AE_{i,j}),\ Sig_{GM}(K_{i,j},\ Enc_{GM}(AE_{i,j})) \qquad \text{for } 1 \leq j \leq N$$

GM stores the value C_i.

4.4.4.3 Signing of message M by A_i (SIGN)

A_i publishes (without any contact with GM):

- M, $Sig(K_{i,j}, M)$
- $K_{i,j}$, $PGK_{i,j} = f_n(C_i, K_{i,1}, \ldots, K_{i,j-1}, K_{i,j+1}, \ldots, K_{i,N})$
- $Enc_{GM}(AE_{i,j})$, $Sig_{GM}(K_{i,j}, Enc_{GM}(AE_{i,j}))$

4.4.4.4 Adding more keys for a group member

Dealt with as the addition of a new group member.

4.4.4.5 Revoking keys

Suppose the GM wants to revoke the keys $K_{i,1}$, $K_{i,2}$, $K_{i,3}$ of member A_i. GM computes and publishes the new group key:

$$Y' = R(b(K_{i,1}), GK) \bmod n$$
$$Y'' = R(b(K_{i,2}), Y') \bmod n$$
$$GK' = R(b(K_{i,3}), Y'') \bmod n$$

and sends to each other participant A_Z

$$Z' = R(b(K_{i,1}), C_z) \bmod n$$
$$Z'' = R(b(K_{i,2}), Z') \bmod n$$
$$C_z' = R(b(K_{i,3}), Z'') \bmod n$$

This calculation produces a different C_z' for each participant A_z. The new value C_z' is to be used by the participants who received it in the production of the signatures, while A_i should not use the keys $K_{i,1}$, $K_{i,2}$, $K_{i,3}$ in the production of the signatures (note the use of the value C_z belonging to each participant in the computation of C_z').

The computation of the roots modulo n resolves in the determination of the inverses modulo $\phi(n)$ of the $b()$'s. Let's call these inverses $c()$'s. Then the value C_z' may be computed as (from property 3):

$$C_z' = C_z^{c(k_{i,1})c(k_{i,2})c(k_{i,3})} \bmod n$$

From property 1, if gcd$(C_z, n) = 1$, then it is possible to compute once

$$s = c(K_{i,1})c(K_{i,2})c(K_{i,3}) \bmod \phi(n)$$
and then
$$C_z' = C_z^s \bmod n$$

for every participant A_z using the proper C_z.

From the latter consideration in property 1 it is possible to have gcd$(C_z, n) = 1$ if gcd$(GK, n) = 1$, as previously requested in the generation of GK (note that C_z is obtained from GK with exponentiations with the inverses modulo $\phi(n)$ of the $b()$'s on the member public keys).

It is easy to see that for every $z \neq i$ (therefore for every participant but the one whose keys were revoked), from

$$GK = f_n(C_z, K_{z,1}, ..., K_{z,N})$$
then
$$GK' = f_n(C_z', K_{z,1}, ..., K_{z,N})$$

but it is not possible to obtain GK' with any combination of A_i's revoked keys, thus those keys cannot be used to compute any signature.

In case the GM is not able to contact a group member to send him the new value C_z', then that group member will use the old C_z. To verify that signature, it is needed the value s previously computed, that may be signed and published by GM in the CRL associated with GK'.

4.4.4.6 Verification by anyone (VERIFY)

To verify the signature produced by any group member the following steps have to be followed:

- verify that $Sig(K_{z,j}, M)$ is the signature of message M using $K_{z,j}$
- verify that $Sig_{GM}(AE_{z,j}, Enc_{GM}(AE_{z,j}))$ is valid.
- verify that $GK' = f_n(PGK_{z,j}, K_{z,j})$ if the signer already used the latest C_z'. Otherwise verify that $GK' = f_n(PGK^s_{z,j} \bmod n), K_{z,j})$ if the signer used C_z. In fact:

$$PGK^s_{z,j} \bmod n = f_n(C_z, K_{z,1}, ..., K_{z,j-1}, K_{z,j+1}, ...K_{z,N})s \bmod n =$$
$$= C_z^{b(k_{z,1})b(k_{z,j-1})...b(k_{z,j+1})b(k_{z,N})s} \bmod n$$

and

$$f_n(PGK^s_{z,j} \bmod n), K_{z,j}) = C_z^{b(k_{z,1})...b(k_{z,N})s} \bmod n =$$
$$= C_z^{s\, b(k_{z,1})...b(k_{z,N})} \bmod n = C_z'^{b(k_{z,1})...b(k_{z,N})} \bmod n =$$
$$= f_n(C_z', K_{z,1}, ..., K_{z,N}) = GK'$$

4.4.4.7 Identification of signer (OPEN)

GM verifies the signature to be opened. Then, using its secret key decrypts $\mathrm{Enc_{GM}(AE_{i,j})}$ and using the fields in the decrypted signatures GM may identify the signer and prove to a third party that A_i signed the public key $\mathrm{K_{i,j}}$ and knows the secret key $K^s_{i,j}$.

4.4.5 Observations and possible improvements

In this paragraph we will discuss some suggestions and possible improvements to our solution. Some of them have been integrated, some are considered as a good subject for future work. Firstly, let’s consider the hashing algorithm described in chapter 4.4.2. When describing it, we went through the question of which is the one with the feature of optimality and security for calculating a one-way accumulator. In this chapter we describe how the original simple function may lead to security issues and how it has been easily improved, as suggested in [ML05]. Some observations are also described for what concerns the OPEN process and also the JOIN process.

4.4.5.1 Observation I

The first observation concerns the hashing function used to produce the one-way accumulator, as described in [BM94] for security reasons.

Specifically, the one-way accumulator function cited is this:

$$e_n(x, y) = x^y \bmod n$$

The suggestion is to modify that function in such way:

$$e_n(x, y) = x^{h(y)} \bmod n$$

where *h()* is a hash function with fixed-length output and opportunely chosen dimension, taking into account that the "exponential" arithmetic is executed in modulo *Φ(n)*.

This variant is aimed at reducing the possibilities of attack due to the representation of the members public keys used for the calculation of the one-way accumulators in the group signature scheme. Indeed, the possibility of implementing such attacks is connected to the kind of representation chosen for those keys. In the following chapter we will provide an example of attack, but they there can easily be found different kind of attacks. It seems wise, therefore, opting for a solution which cuts out the vulnerability since the beginning, instead of being forced to run security analysis for each implementation.

4.4.5.2 Example of attack

Let's suppose, for simplicity, that $N = m = 3$. Let's suppose also that as public key cryptosystem be used the discrete logarithm on Z_p, with p of type *strong prime*. The member's keys become then:

$$k_{i,j} = g^{k_{i,j}^S} \bmod p$$

Let's suppose that members A$_1$ and A$_3$ are conspiring, therefore they generate their keys in this way:

$$k_{1,1} = g^{x_1+x_2}, k_{1,2} = g^{k^S_{1,2}}, k_{1,3} = g^{k^S_{1,3}}$$
$$k_{3,1} = g^{x_1}, k_{3,2} = g^{k^S_{3,2}}, k_{3,3} = g^{k^S_{3,3}}$$

Where $g^{x_i} < \sqrt{p}$ and the various exponentiations are meant *mod p*. Let's suppose now that member A$_3$ is revoked. He aims now at being able to issue valid signatures in spite of the revocation. The GM (Group Manager), after the revocation of member A$_3$, updates the parameters and sends them to A$_1$:

$$C_1 = f(X, K_{2,1}, K_{2,2}, K_{2,3})$$

A$_2$ receives, differently, this parameter:

$$C_2 = f(X, K_{1,1}, K_{1,2}, K_{1,3})$$

while the new group public key is updated by signing the value:

$$GK = f(X, K_{1,1}, K_{1,2}, K_{1,3}, K_{2,1}, K_{2,2}, K_{2,3})$$

Now, let's suppose that A$_3$, in spite of the revocation, wants to sign with the key K$_{3,1}$ the message M. To do so, A$_3$ calculates (with C$_1$ provided by member A$_1$):

$$M, Sig_{K^S_{3,1}}(M), K_{3,1} PGK_{3,1} = f(C_1, g^{x_2}, K_{1,2}, K_{1,3})$$
$$Enc(R_1,3), Sig_{GM}(K_{3,1}, Enc(R_1,3))$$

Notice, in fact, that:

$$PGK_{3,1} = f(C_1, g^{x_2}, K_{1,2}, K_{1,3}) =$$
$$= f(X, K_{2,1}, K_{2,2}, K_{2,3}, g^{x_2}, K_{1,2}, K_{1,3})$$

from hence:

$$\begin{aligned}
&f(PGK_{3,1}, K_{3,1}) = \\
&= f(X, K_{2,1}, K_{2,2}, K_{2,3}, g^{x_2}, K_{1,2}, K_{1,3}, g^{x_1}) = \\
&= f(X, K_{2,1}, K_{2,2}, K_{2,3}, g^{x_2+x_1}, K_{1,2}, K_{1,3}) = \\
&= f(X, K_{2,1}, K_{2,2}, K_{2,3}, K_{1,1}, K_{1,2}, K_{1,3}) = GK
\end{aligned}$$

Conclusion: here is demonstrated the necessity of preventing such attacks by representing the public keys in a different way. That's why it's suggested the use of a hash function in the e_n function. The above illustrated attack is feasible when it's possible to generate two private keys whose corresponding public keys may factorize. The example with the discrete logarithm shows how the overall security of the whole system is lower than expected (it's sufficient finding some x such that $g^x \bmod p < \sqrt{p}$ to be able to generate any factorization of public key, selecting, for instance, couples of keys $x1 = x - a$, $x2 = x + a$). It's possible that there exist other circumstances on p and n by which the attack can be carried on, in addition to further representations as well. This is one more reason why we decided to modify the function e_n, in order to break the possibility of factorization of public keys.

4.4.5.3 Observation II

In order to render the function OPEN more effective, it's suggested the insertion in the signature of each group member some kind of information which connects the signature itself to the member who issued the signature. This information should be shown to a third party in case of dispute, and the third party should be convinced by the correctness of such data. In this direction, the set of information:

$$Enc(R_j, i), Sig_{GM}(K_{i,j}, Enc(R_j, i))$$

ought to make use of R_j different from random data. For instance, it could be the signature of the member i who asks for the insertion into the one-way accumulator, and then into the public key of the group, of his own key $K_{i,j}$. Such request must, in fact, prove also the real possession of the corresponding private key $K^S_{i,j}$.

The *non-traceability* may be preserved by means of a symmetrical algorithm, for instance in CBC mode with random IV.

Conclusion: the opening of the signature may reveal the real identity of the member who signed by means of some data which can be used to persuade a third party.

4.4.5.4 Observation III

The Group Manager (GM) should provide each member with the list of signed keys, in a way similar to what is already known as *certificate*. The hash accumulator should provide the system with a more effective bandwidth and speed benefit, besides the unpleasant event of keys exhaustion, event which could possibly lead to DoS attacks.

The best scenario is that in which the members need not be reached by the GM, or get in touch with him, for each group adjustment operation. For instance, there could be some benefits trying to limit the connection with the GM for every JOIN, as described in a rough and intuitive way in the following. Notice that it's just an idea and not a well defined schema, but it could lead to some notable improvements.

4.4.5.5 New JOIN

Let's assume that when the system is up and running there are only 3 group members. The GM calculates:

$$GK = f(X, K'_{1,1}, K'_{1,2}, K'_{1,3}, K'_{2,1}, K'_{2,2}, K'_{2,3}, K'_{3,1}, K'_{3,2}, K'_{3,3})$$

Join member A_1. The member generates 3 public keys $K_{1,1}$, $K_{1,2}$, $K_{1,3}$ and sends the signed requests to the GM, receiving back from him:

$$C_1 = f(X, K'_{1,1}/K_{1,1}, K'_{1,2}/K_{1,2}, K'_{1,3}/K_{1,3}, K'_{2,1}, K'_{2,2}, K'_{2,3}, K'_{3,1}, K'_{3,2}, K'_{3,3})$$
$$Sig_{GM}(K_{1,j}, Enc(R_j, 1))$$

with j = 1, 2, 3.

Join member A_2. The member generates 3 public keys $K_{2,1}$, $K_{2,2}$, $K_{2,3}$ and sends the signed requests to the GM, receiving back from him:

$$C_2 = f(X, K'_{1,1}, K'_{1,2}, K'_{1,3}, K'_{2,1}/K_{2,1}, K'_{2,2}/K_{2,2}, K'_{2,3}/K_{2,3}, K'_{3,1}, K'_{3,2}, K'_{3,3})$$

$$Sig_{GM}(K_{1,j}, Enc(R_j, 1))$$

In this formula, j = 1, 2, 3. In this approach there's no need to provide a new C_1 to the member A_1. Actually, this method requires that the GM be able to calculate $K_{i,j}^{-1} \bmod \phi(n)$. The GM only knows the factorization of n, then it's the only entity who is able to compute *Φ(n)*. However, in order to do so, it's necessary that $\gcd(K_{i,j}, \phi(n)) = 1$. For instance, the member who is joining the group could send his keys to the GM, and the latter could select and sign only those prime with *Φ(n)*. Notice that if *n* is a rigid integer (as suggested in paper [BM94] together with the definition of one-way accumulator) then n = pq, where *p = 2p'+1* and *q = 2q'+1* with *p'* and *q' safe primes*, in addiction to distinct. In this situation, in order to apply the protocol *New JOIN*, it's sufficient that the public keys generated by the members (or their hashes if the variant described above is applied) are not divisible by 2, *p'* and *q'*, an easily verified condition.

Conclusion: some features of the discussed method can be improved, such as the necessity of being reached by a PUSH at every group adjustment, and the possibility of being victim of DoS attacks based on keys exhaustion. The observation above proposed may therefore be evaluated (in terms of correctness, security and feasibility) in order to try to reduce the necessity of communication with the GM for each group modification.

Chapter 5 Implementation

5.1 Introduction

The method used by Assolo to store the network traffic towards the administered systems has been presented. The stored data are non-repudiable by the system administrator. It arises the necessity to store these data (which are essentially a log file) in a secure format. That is, the logged data should be stored in such a way that maintains the privacy of the administrators and, at the same time, cannot be modified/erased without notice.

Since January 2004, we have worked in collaboration with Telecom Italia Labs in order to find an effective solution to the issue of *irrefutable administration,* and consequently non repudiation of data, described in the previous sections. We used part of the research contained in this volume in order to develop such system, where commands issued by system administrators are securely archived and only some auditors are, in certain conditions, able to verify what the log contain. The developed system contains the following features:

- Non-repudiation: the administrators must not be able to refuse to acknowledge the contents, the order and the time in which data and commands have been issued to the administered system; this must be true for he who sends but also for he who receives data;
- Anonymity: the administrators must be able to hide their identity so as to protect their right to privacy; at the same time, on the other hand, it must be possible, in case of dispute and only by means of authorized personnel, to reveal the identity of the administrator and the log of his work;

These two features are obtained by means of cryptographic system, which include among the others one-time signatures, group signature, pseudonyms and receipts.

5.1.1 Assolo

Assolo [MI04] is the name of the system developed in collaboration with Telecom Italia Labs to grant the non-repudiation of sent/received data. It. While most of the network forensics tools nowadays (e.g. Infinistream Forensics Security or NetIntercept) act as sniffers on the network, Assolo acts as a gateway on the network. When a user wants to connect to an administered system, his connection is redirecter through Assolo, which takes care, in turn, to connect to the final endpoint. The first benefit of this approach is that Assolo is able to store all the data traveling on the network, no packet is lost. At the same time, on the other hand, gateways introduce some issues on some well-known protocols, such as FTP, which have to be used in different ways.

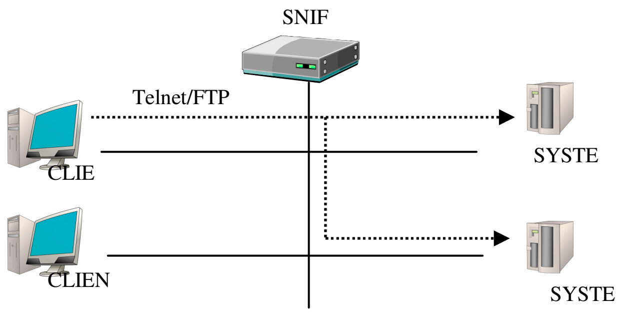

Figure 5-1: Sniffer-based architecture

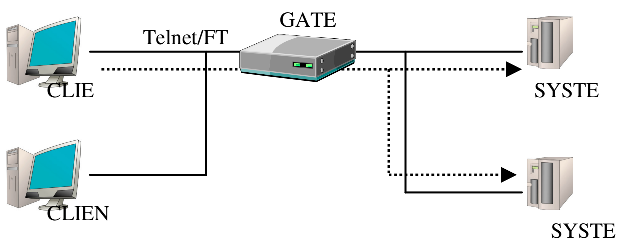

Figure 5-2: Gateway-based architecture

In order to grant the non-repudiation of data, the system makes use of one-time signatures and receipt, building on that a protocol which is inserted into the TCP/IP stack, between the TCP and the application level. More precisely, the whole protocol is encapsulated inside a SSH tunnel, in order to benefit from the encrypted data channel SSH builds as part of the protocol.

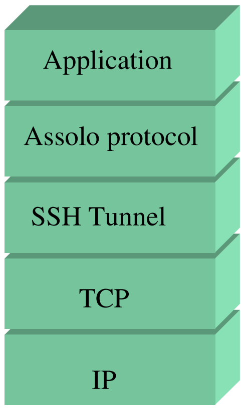

Figure 5-3: Protocols stack

As mentioned before, Assolo stores all packets traveling on the network through its gateway. More precisely, it stores on memory devices the SSH tunnel payload, that is the Assolo protocol and the application protocol. The archived data can be therefore analyzed or kept for a-posteriori inquiries, for that reason the application protocols must not be encrypted (for instance telnet or ftp).

In case of encrypted application protocols, there is the need of some key escrow or key recovery methods, or yet encryptions made by third trusted parties, in order to be able to decrypt the archived data a-posteriori. Those approaches have been discarded because of the difficult infrastructure they require and of the risks they can lead to; in [AAB98] the limits connected with this infrastructure are discussed in details.

The communication between the client (i.e. the administrator) and Assolo is kept encrypted by the aforementioned SSH tunnel. In order to encrypt also the communication between Assolo and the administered server we need a further step. The user who wants to get connected to a server through the SSH protocol will, therefore, have to connect to a known port on the Assolo server via telnet, hence he will access a shell where he will specify the endpoint he wants to securely connect to.

5.2 Software architecture of the system

In the previous chapters some methods and principles have been shown that together can take part in a secure data storage with group access privileges and high privacy enforcement. In this section we will describe the implementation of such system, developed in collaboration with the Computer Science Department of Turin and Telecom Italia Labs in 2004. Some decisions have been taken in order to make the system as much efficient and effective as possible, keeping into consideration the constraints of a real implementation.

In particular, we will describe the system architecture, the modules which concur to give strength to the whole solution, the single features and how stored data are archived into a database.

This project has been called "*Assolo*", and this name denotes also the module which contains the Log Manager Interface and which is in charge of capturing data from administrators/systems transactions, as we will see in the following.

The secure archiving system we are designing, which will be called *Log Manager*, is in charge of communicating with the Assolo module and to store all the data the latter captured during his job.

As described in chapter 5.1.1 Assolo becomes a sort of gateway between the remote administered computer and the administrator, and its operation is that of capturing all the packets traveling between those two entities in order to keep trace of what it's being going on. In a second place or time, therefore, it will be possible to check the correctness of the operations which the administrator has issued on the remote computer. This means that in case of dispute – in which the administrator could be the prosecutor or the prosecuted – there will be a secure and certified data stream to be opened and used as probation for the cause.

The whole of the captured data is placed into temporary files sent by Assolo to the Log Manager, in order for the latter to take the charge of encrypting and securely storing the data. This transmission is made using the TCP/IP protocol, and we can put forward that it will be required to attach some context information useful to the Log Manager for a correct storing process.

The Log Manager, after the encryption of data, takes the charge of sending those data to a DBMS; in the following we will discuss the reasons why a database becomes necessary to the storing process and what is the relational scheme of the tables which will be used.

Notice that the DBMS must be reachable also from other remote systems, in particular from the auditors' workstations. The auditors will, in fact, be able to issue queries and retrieve archived transactions data. For this reason it's necessary to introduce some techniques to distinguish the privileges of the single users, considering also the Log Manager and the DBMS administrator, who must be able to manage the configuration by means of some tools designed for the purpose.

Assolo, Log Manager and DMBS must not necessarily be installed on the same physical computer, but it's desirable that between those entities there exist some secure and single-purpose connections. In particular, the data traveling between Assolo and Log Manger are not encrypted, therefore it's necessary to prevent possible attacks based on packet sniffing.

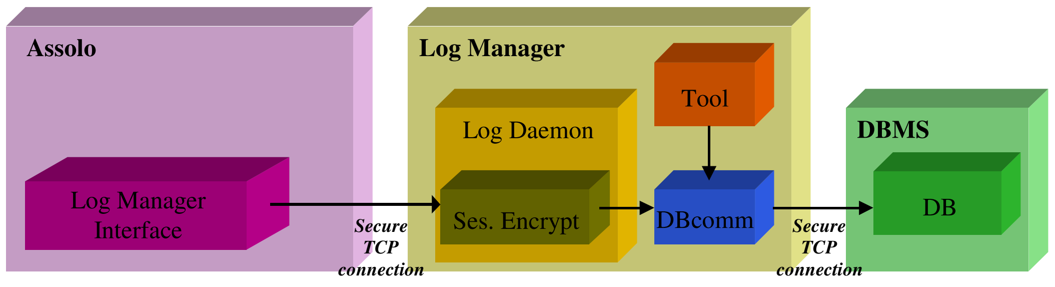

Figure 5-4: System architecture

As shown in the figure, the Log Manager is made up of several logical modules, more in details:

- Log Daemon, which waits for the data from Assolo;
- Session Encrypt, which attends to encrypt those data and to prepare the secrets for the auditors (i.e. group members);
- DBcomm, which is in charge of the communication with the DBMS, of the signature of the inserted data and of the authenticity check of the received data;
- Other tools, which represent the interfaces provided to the Log Manager administrator to configure the auditor's personal data, archived in the database as well;

5.3 Communication between Assolo and Log Manager

In this section we will analyze the data Assolo sends to the Log Manager and the size and format they must obey to.

Assolo takes the role of an intermediate entity between the administered element and the administrator, and deals with capturing all the IP packets (belonging to a whole work session) traveling between them. In particular, it has to reassemble in a single bit stream the packets payload relative to a whole session, and then send this stream via TCP/IP to the Log Manager.

Attached to the stream are the data characterizing the work session, denoted in Chapter 3 simply with U_h. We can now list with more details the data which concur to identify a whole session:

- type of session (correctly concluded session, not correctly concluded session)
- IP address of the administered server
- applicative protocol used for the administration (it can be deduced by the port the connection to the administered server has been addressed to)
- administrator pseudonym
- work session beginning and ending time

Furthermore, Assolo is also engaged in the task of preparing the list of the auditors and of the groups which are entitled to access a given record of information. That list is created by means of three pieces of information: the administered server, the real identity of the operator/administrator and the type of protocol used in the administration process. Once this list has been assembled, Assolo sends it to the Log Manager.

The format of the packet Assolo uses to send data to the Log Manager is the following:

0	8	16	24	31
type	auditor_num	port_ae		
ip_ae				
nym_id				
start_time				
end_time				
session_num				
Auditor				

Figure 5.4: Format of the packets Assolo uses to send data to the Log Manger

The fields involved in this packet are the following:

- type (8): type of session (normal, bad);
- auditor_num (8): number of auditors and groups entitled to acces the data (this value multiplied for 16 gives the length in bit of the *auditor* field);
- port_ae (16): TCP port used for the administration of the remote system; this port gives a hint on the protocol used for the administration;
- ip_ae (32): IP address of the administered system;
- nym_id (32): administrator pseudonym;
- start_time (32): exact time identifying the beginning of the session;
- end_time (32): exact time identifying the ending of the session;
- session_num (32): session number; it's a numerical unique identification Assolo assigns to each work session of the administrators:
- auditor (variable length): list of the auditors and of the groups who are entitled to access the recorded data; both auditors and groups are unambiguously identified by 16 bits codes, where the first bit gives the discrimination between auditors and groups;

Notice that the data which must be actually recorded is not present in the payload of the packet Assolo sends to the Log Manager. The explanation is that the temporary files Assolo employs to save captured packets in are stored in an area where the Log Manager has read access rights. The Log Manager is, therefore, able to retrieve autonomously the file whose name is identified by the hexadecimal codification of the session number and whose location is fixed.

The file Assolo assembles does not contain the whole IP or TPC captured packets, but only the data payload encapsulated in the SSH tunnel, as shown in figure:

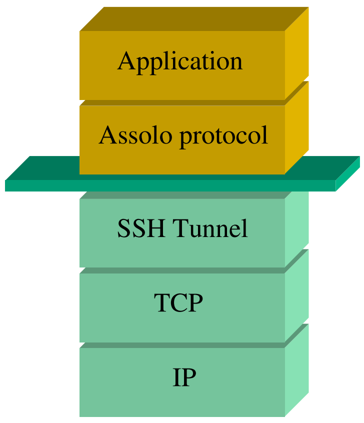

Figure 5-5: Protocols stack pointing out what is stored

More precisely, the payloads of all the packets are chained in a whole data flow. It's not necessary to introduce, when saving the data, specific headers to the file or to the single packets, since all the information which is necessary to the auditing process is already included in the packets header of the Assolo protocol.

5.4 Selected algorithms

In Chapter 2 we introduced several algorithms concerning symmetrical and asymmetrical encryption, secret sharing and hashing functions, and in Chapter 3 we described how to use them to develop a secure storage facility.

We did not specify, however, which among all those possible alternatives we would choose for the implementation. Some observations must be made on the reasons which have led to the choice of particular methods among the ones we had at disposal.

Notice furthermore that, among the two alternatives described in paragraphs 2.3 and 2.4 (symmetrical or asymmetrical keys for each auditor) we preferred the asymmetrical solution. That's because by means of public key encryption systems there is no need of keeping a shared secret (i.e. the symmetrical key) in the Log Manager area, providing more security mainly in case of system attacks.

The various features of the designed system require the use of symmetrical and asymmetrical algorithms, hash functions, secret sharing methods, random numbers generators, text-binary conversion functions and data compression solutions.

In paragraph 3.3 and following, we stated the necessity of using – to encrypt the file to be stored – a symmetrical key with length equal to the hash output.

Consequently, we need to find an symmetrical encryption algorithm with a key dimension equal to the length of the digest issued by a hashing function. The reason of this choice is the uniformity of structure: it's advisable to keep a well defined structure for the secrets belonging to each auditor, because the data to be encoded is simply made of bit strings, without headers or formatting information.

We want to provide a high security level for the designed system, therefore much more consideration has been laid on the security features of the chosen algorithm than on their performance values.

For this reason we evaluated the symmetrical keys as long as the digest produced by MD5 or SHA-1 (respectively 128 and 160 bits) not sufficient. On the other hand, the algorithm chosen for the symmetrical encryption is AES because it works on blocks of 128 bit (although Blowfish would have had higher performance). In conclusion, the chosen algorithms are SHA-256 and AES – the latter may adopt 256 bits keys.

Those features allow for a good level of security also for the non-immediate future. This is particularly relevant if we consider that a dangerous attack to cryptosystems is the possibility of decrypting, in a future when computational power will be much higher than now, data encrypted at present.

For analogous reasons, we chose asymmetrical 2048 bits keys instead of the more common 1024 bits keys; some researchers (e.g. H. H. Orman e P. Hoffman in http://www.ietf.org/internet-drafts/draft-orman-public-key-lengths-08.txt), assert that RSA-1024 is less safe than a symmetrical cryptosystem with 80 bits keys.

As for the random number generation, we evaluated the method described in ANSI X9.17 appendix C as quite good: this algorithm, in fact, states optimal distribution properties and independence between the generated values, and it's widely used in many common cryptographic applications.

As before discussed, Shamir's solution has been adopted when it comes to secret sharing: it combines simplicity, efficiency and reliability. From what we illustrated in the previous sections it should be clear how this solution may be optimal in terms of features for the designed system.

In conclusion, for data compression the Gzip algorithm has been chosen, while for the binary-text-binary conversion the Base64 method seemed to be well suitable for the designed system.

5.5 Cryptographic libraries: Crypto++

The implementation of the designed system required a library which contained all of the functions involved in the process: symmetrical and asymmetrical encryption, hash functions, random numbers generator, etc… In particular, those libraries had to implement the algorithms listed in the previous sections.

Several cryptographic [CE04] libraries are available in the public domain – the open source community has released a lot of code concerning cryptography – but the most known are the OpenSSL (http://www.openssl.org/) libraries. They are largely used in a wide number of applications concerning signatures and data encryption: they include hash and MAC functions, symmetrical and asymmetrical encryption algorithms, data compression and conversion code.

As for the implementation of the system, however, those libraries are limited in that they include neither secret sharing functions nor an implementation of SHA-256; as stated in the previous section, those two methods are necessary for the implementation of the designed system.

For that reason, we decided to use a different implementation of cryptographic libraries, open source as well, offering high levels of security standards. We found the Crypto++ (http://www.cryptopp.com/) to be a good candidate. They are less known than OpenSSL, but efficient and complete to the same extent. The programming language used for those libraries is the C++, and they include most of the algorithms we need for the development: symmetrical and asymmetrical encryption, signatures, hash and random numbers generator functions. In those libraries we find also an implementation of Shamir's secret sharing method described [SHA79], method more than indispensable for the correct operation of the system.

The completeness of those libraries is such that they include also data compression functions – such as Gzip and Deflate – and data conversion methods – for instance Base64; those implementations are very useful during the development process, as described in the previous section.

Another reason why we chose the Crypto++ libraries is that a version of those libraries has been certified by NIST: Crypto++ were compliant to the specifications stated in FIPS 140-2. Since NIST does not certificate source code, but only binary code or howsoever pre-compiled modules, the only version to be certified is the 5.0.4. More precisely, NIST has certified a DLL library, available only for Microsoft OS, which does not include all those algorithm that NIST retains unnecessary for a cryptographic library. Crypto++ include, indeed, a lot of functions and variations of well known methods widely used. It's possible, during the compilation process, to include all of those functionalities (thus compiling the standard version, the one with all the libraries available) or to exclude them, obtaining a library with the functions certified by NIST and nothing more.

Notice, however, that when the library is compiled in the autonomous way it cannot be considered as certified. In the official homepage there is, on the other hand, the DLL version that NIST considered compliant to FIPS 140-2.

In order to get the code to run both in Windows and in Unix-like (e.g. Linux) OSs we had to compile the libraries, thus losing the official NIST certification. Nevertheless, the code may be regarded as safe, at least the part of the code which implements the functions NIST certified.

5.6 Log Manager functionalities

In this chapter we will describe the functionalities implemented by the Log Manager, focusing our attention on its tasks and their sequence. In particular, we will concentrate on the three more relevant functionalities of the Log Manager, that is:

- Session storage
- System initialization
- System halt

5.6.1 Session storage

The Log Manager is made up by three main modules: Log Daemon, Session Encrypt and DBcomm. They have different specific tasks and, more precisely, all of them output the results of their task to the following module in the chain.

As for the new session storing process, the first module involved is the Log Daemon: it waits on a TCP port for the data to be stored and behaves like a common daemon process. In details, the Log Manager itself is a multi-thread program, therefore the Log Daemon, upon receiving a TCP connection, starts a new ‘child’ thread for the management of that connection. In the meantime, the ‘parent’ thread stays on waiting for new connections on the same port.

One of the tasks of the ‘parent’ thread is that of keeping in a shared memory area with semaphores access, the last ring of the hash chain which has been used. In the following we will discuss the reason which justifies such a behavior.

The 'child' thread receives the data from Assolo and stores them on a temporary file, which will be deleted upon their correct archiving on the DBMS. Therefore it invokes the Session Encrypt module, which takes in input those data.

The Session Encrypt module is then in charge of checking whether there exists a file whose name is the hexadecimal codification of the session number: this file must contain the data received from Assolo and exist in the shared memory area. If that file exists, the module will proceed with the generation of the secrets for each auditor. If the file is not present, it will send back to Assolo an error message stating that the file could not be found.

The Session Encrypt must now retrieve from the DB the information about the active auditors; this is done through the collaboration of the DBcomm which, upon executing the queries and retrieving the required information, frames those data in an agreed upon structure.

At this point the Session Encrypt can generate with a pseudo-random algorithm the session key A_i and calculate its hash $H(A_i)$. Using the data received from Assolo, in particular the list of auditors and groups entitled to access data, the module can calculate the secrets relative to each auditor and then encrypt them with the corresponding public key.

More precisely, the module will calculate the sub-secrets belonging to each auditor of a group, then he will concatenate them with the key – real or fake – assigned to each one of them, obtaining the structure described in paragraph 3.6.

If an auditor does not belong to any group, or he belongs only to groups not entitled to access that particular record, the information relative to those groups is omitted, because it's not necessary (to assure the elusion property) to assign fake data to these auditors.

In order to allow an auditor to be part of more than one group (potentially infinite groups), during the implementation process we chose to let each auditor have more than just one encrypted block in the record. More precisely, a configuration file is used to set the number of blocks RSA-2048 destined to each auditor: each auditor can belong to a number of groups equal to 5 multiplied for the number of blocks he's been assigned to. This value is computed considering the decryption key length its hash length (256 bits) and considering the length of each sub-secret (336 bit). Notice that all the auditors must take the same number of blocks, also those who wouldn't need such amount of space, in order to enforce the elusion property. Without this 'trick' it would be, in fact, easy to tell how many groups an auditor is in simply by analyzing the encrypted data.

Let's recap: the module computes the sub-secret s, it concatenates them to the real or fake key, and it encrypts them with the public keys of all the active auditors. Optionally, data can now be packed to reduce their size. This step is optional because it could, in some cases, affect the overall system performance and should then be avoided. The algorithm by default for data compression is Gzip, but it's easy modifying it by editing the Log Manager configuration file.

Such data is now ready to be symmetrically encrypted with the key A_i. The encryption step should always be subsequent to the compression, because the encryption process removes all the structures and regularities present in a file thus preventing the compression from reducing the effective size of the file. The reason is that compression is often based on regularities and repetitions in the file; encryption, on the other hand, makes the file much similar to random data stream.

The file is now archived in a storage area both Log Manager and DBMS have access to, while in the DBMS tables are stored the path and hash of the file itself.

Notice that the implementations is slightly different from the process described in Chapter 3. For efficiency reasons, in fact, it was chosen not to insert the encrypted data straight into a BLOB field (Binary Large Object) of the table containing the logs. It was decided, on the other hand, to store only the file path and its hash. Consequently, the method used to calculate each ring of the hash chain has to be modified as well: in this computation the system will not make use of the whole file but only of its path and hash. This slight modification does not affects the security of the whole process, its only aim is that of avoiding to store big sized data straight into a DBMS table.

At this point the Session Encrypt can compute the new ring of the hash chain. More precisely, the hash will be computed on the concatenation of the following data:

- Session identification data (elements which can be used to identify correctly the session);
- Secrets assigned to each auditor;
- File session path and hash;
- Previous ring of the hash chain;

The last ring of the hash chain correctly archived in the DBMS is stored in the shared memory area (whose access is governed by semaphores) previously described.

The lapse of time occurring between the Session Encrypt access to this area and the confirmation of the DBcomm about the correctness of the session data storing process is contained in a critical section. The other executing thread will therefore be suspended when trying to get access to the last ring of the hash chain.

The Session Encrypt, upon receiving the DBcomm confirmation about the correct data insertion, will update the value of the last ring of the hash chain so as to make it available for the waiting threads.

As discussed before, the Session Encrypt sends the DBcomm the data to be inserted into the DBMS, which in turn will take care of the insertion and of the signature of the data received (this signature will be issued with the private key of the system itself).

Notice that the DBcomm, upon receiving data from the database, takes care of checking the authenticity of the signature as well.

After the correct insertion of the received data into the database, the Session Encrypt proceeds deleting the temporary file created the Log Daemon, containing the data necessary for the encryption and for the identification of the archived data, and the temporary file created by Assolo, containing the unencrypted data to be archived.

5.6.2 System initialization

During the system initialization some operations must be carried on: firstly, the system must check the DBMS activity (it must be active and reachable in order to receive the data), then it must check that the tables where data will be archived exist and are built according to the expected structure. The system must subsequently check whether the table containing log files is filled with data, and in case getting the last ring of the hash chain correctly stored. This hash value will be stored in the shared memory area previously described, and will be used as starting value for the following rings computed on data to be archived.

It's also possible that, for unpredictable reasons, the Log Manager crashes and stops working. That's why, after an unpredicted stop, the Log Manager must check that all the files Assolo sent are correctly stored into the DBMS. In order to do that the Log Manager must check whether the temporary files, containing the data sent by Assolo and concerning a whole session, have already been used to store data. This check can be carried on by searching into the DBMS for the sessions concerning those files. If they can be found the temporary files are deleted, otherwise the files will be passed as input to the Session Encrypt module which will then take care of retrieving the session file from the memory area shared with Assolo and proceed with encrypting and inserting data into the DBMS as described in the previous sections.

After those operations have been carried on, the system will open again the TCP port where it will listen for connections coming from Assolo.

5.6.3 System stop

It’s important being able to plan a correct stop to the archiving system, because it may turn out to be necessary sometimes: for instance in case of maintenance.

In order to terminate correctly, the Log Manger must firstly not accept connections coming from Assolo. In this way he will not receive further data to be archived, encoded and sent to the DBMS.

Next, the system will have to wait for the correct termination of all the encryption processes occurring at the time the stop command was issued; by doing so no computed data will be lost.

Only after having completed correctly such procedures, the Log Manager execution can be safely interrupted.

5.7 DBMS data storage

In this section we will describe the reasons and the benefits of the employment of a DBMS for data storage. We will include in the description also the relational scheme of the tables used for the storage of logs and of the data concerning groups and auditors.

5.7.1 Description of choices

The system we are describing, as afore illustrated, stores data into a DBMS. The main reason for this is the need for an easy to use and to access archiving system, crash resistant and with features such as stability and transactional support (for instance correct termination of writing operations).

The possibility of using a simple file managed directly by the Log Manager has been considered but eventually discarded, firstly because the file structure would have been too complex. A database, on the contrary, allows for quick and easy queries on archived data (i.e. the queries issued by auditors), mostly when indexes and data caching techniques are used to improve system performances.

Furthermore, the DBMS supports privilege, therefore it’s quite easy assigning users different privileges according to their role and managing automatically the concurrent access of several users.

Quite a few DBMS systems are available on the market, each one with its several features and its different costs. Oracle is, by far, one of the most reliable and complete, mainly when it comes to security; unluckily, the cost for each license is too high. Therefore, when the target is the first implementation of a new system, most organizations adopt products with good features but lower costs.

For the implementation of this system, which may at this stage be easily regarded as a prototype, we chose to adopt an open source DBMS, which obviously has no costs but can offer lots of necessary functionalities all the same.

Among the different open source DBMS available on the public domain, MySQL was the one we decided to adopt for the prototype of the system. MySQL is widely used, stable, reliable and well tested with the help of years of practice and thousands of users all around the world. As for performance, MySQL is one of the best choices, both because it's been around for some years and therefore it's been refined and because it doesn't offer, in its standard distribution version, some of the typical advanced DBMS features. In particular, the default installation does not allow for referential integrity rules (foreign keys) and cannot make use of the protocol of commit/rollback for the queries.

However, those functionalities can be activated by editing the system configuration file, but in this case system performances are negatively affected. During the implementation, we decided to make use of foreign keys for tables relationships all the same, while we didn't evaluate as necessary the commit/rollback feature.

5.7.2 Relational schema

In the following are described the tables which will be stored in the DBMS, starting from the tables and relationships scheme.

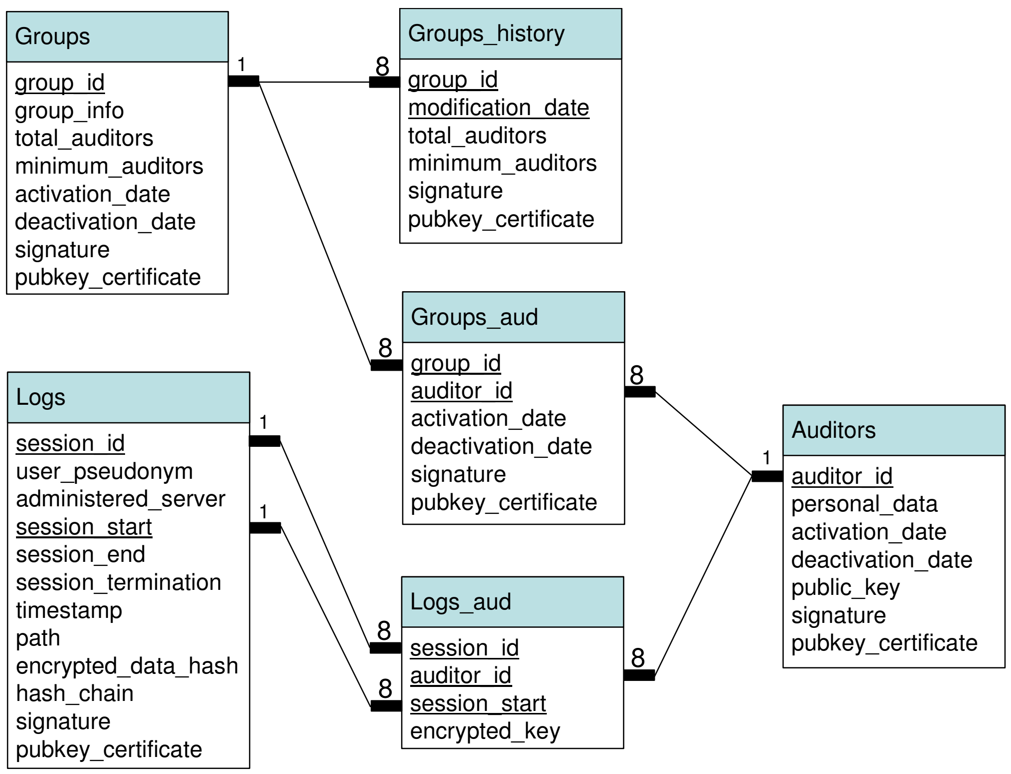

Figure 5-6: Relational schema of the DBMS tables

In the figure, the underlined attributes represent the primary keys of the tables, while the lines which link together the tables stand for the referential integrity rules with the corresponding cardinalities.

In the following we will detail the tables showed in the figure.

5.7.3 Table details

The first table described here is the one containing the data concerning the auditors: the "Auditors" table. Its attributes have the following format and meaning:

- **auditor_id**: it's a 16 bit code, whose first bit vale is equal to 0; it unambiguously identifies each auditor and is the primary key of the table;
- **personal_data**: this attribute contains the auditors' personal information: name, surname, address, tax code; the format doesn't affect the behavior of the Log Manager, to the extent that this field could also be empty or contain lots of data;

- **activation_date**: it contains the date in which the auditor has received the access grant to the logs;
- **deactivation_date**: it contains the date in which the auditor has been revoked the access to the logs; when the auditor has not been revoked, this field contains a NULL value, which means a not valid date;
- **public_key**: this attribute contains the auditor's certificate; the certificate contains, in turn, the public key of the auditor:
- **signature**: this field contains the digital signature which DBcomm issues to data when they are inserted of modified;
- **pubkey_certificate**: it contains the certificate of the public key used to create the signature of the previous field;

The attributes of the table "Groups" have the following format and meaning:

- **group_id**: this is a 16 bit value, whose first bit is equal to 1; it unambiguously identifies a group and is the primary key of the table;
- **group_info**: this attribute contains all of the data characterizing a group; as it was for the "personal_data" attribute, its format or size does not affect the behavior of the Log Manager;
- **total_auditors**: this field contains the number of auditors belonging to a specific group in that moment; if the value of this field is modified, a new record must be added containing the old value in the table "Groups_history"; the reasons of this addition are detailed in the following.
- **minimum_auditors**: it contains the minimum number of auditors belonging to a group whose collaboration is necessary to decrypt the record (the data contained in the record); if a modification to this field occurs, the same behavior as the preceding field is applied;
- **activation_date**: it's the date when the group was built;

- **deactivation_date**: it contains the date since when the group is not allowed anymore to have access to the log; if the group is still entitled to access the data, this field contains a NULL value;
- **signature**: it contains the digital signature DBcomm issues on data when they are inserted or modified;
- **pubkey_certificate**: it contains the certificate of the public key which has been used to create the signature contained in the preceding field;

In the table “Groups_aud” there is the list of the auditors who are members of each group; the key of this table is composed by the couple group_id and auditor_id which, on top of that, must obey to the foreign key rule with the homonymic fields of the tables “Groups” and “Auditors”. The attributes of this table have the following structure and meaning:

- **group_id**: as this field is the foreign key for the table, it submits to the same rules of the homonymic field in the “Groups” table;
- **auditor_id**: as in the previous item, this attribute is a foreign key, therefore it has got the same structure as its homonymic field in the “Auditors” table;
- **activation_date**: it contains the date in which the auditor, identified by the “auditor_id” value, entered the group identified by “group_id”;
- **deactivation_date**: it contains the date in which the auditor, identified by the “auditor_id” value, was revoked from the group identified by “group_id”;
- **signature**: this field contains the digital signature the DBcomm issues on data when they are inserted or modified;
- **pubkey_certificate**: it contains the certificate of the public key used in the signature contained in the previous field.

The table “Groups_history” contains the historical list of changes which occurred to the table “Groups”. More precisely, are kept in this table the changes occurred to the attributes “total_auditors” and “minimum_auditors”. Those data must be kept because, as discussed in Chapter 3 the changes of these values can’t have retroactive validity. Consequently, when during the auditing process an auditor wants to access data *as member of a group*, the system must know whether the user was member of that group when the record was inserted in the database.

The key of this table is made up by the two attributes “group_id” and “modification_date”. The format and meaning of the attributes of this table are the following:

- **group_id**: this field represents the foreign key of the table, therefore it’s similar to the homonym field in the “Groups” table;
- **modification_date**: it contains the date in which the parameters concerning a group have been modified;
- **total_auditors**: it contains the number of the auditors belonging to the group when the modification (see previous field) took place;
- **minimum_auditors**: it contains the minimum number of auditors – belonging to the group when the modification took place – whose collaboration is necessary to access to the archived data;
- **signature**: this field contains the digital signature the DBcomm issues on data when they are inserted or modified;
- **pubkey_certificate**: it contains the certificate of the public key used in the signature contained in the previous field.

The table “Logs” maintains the data concerning each archived session; its primary key is composed by the attributes “session_id” and “session_start”. Session identifiers are randomly generated and it’s highly improbable, though still possible, that two session identifiers assume the same value. The attributes of the value have the following format and meaning:

- **session_id**: it’s a 32 bit code which, together with the field “session_start”, identifies unambiguously the work session;
- **user_pseudonym**: it’s the administrator’s pseudonym relative to the session;

- **administered_server**: this field contains unambiguously the administered server and the administration protocol used; in general it coincides with the IP address of the administered calculator and the TCP port used;
- **session_start**: it contains the time in which the work session started and, together with the field "session_id", identifies unambiguously the work session;
- **session_end**: it contains the time in which the session was closed;
- **session_termination**: this field signals whether the session has been correctly closed or there have been anomalies;
- **timestamp**: this attribute, typically administrated automatically by the DBMS, contains the time in which the data concerning a defined session have been inserted into the table;
- **path**: it represents the logical path where the encrypted file containing the packets concerning a session is stored;
- **encrypted_data_hash**: in this field there is the hash of the encrypted file containing the packets concerning a session;
- **hash_chain**: this attribute contains the hash chain computed as shown in paragraph 5.6.1;
- **signature**: this field contains the digital signature the DBcomm issus on data when they are inserted or modified;
- **pubkey_certificate**: it contains the certificate of the public key used in the signature contained in the previous field.

In conclusion, the table "Logs_aud" is that which contains the secrets pertaining to each auditor, useful during the session decryption process. More precisely, whenever a session is stored in the table "Logs" previously discussed, in this table for each active auditor a record is inserted. This record contains the data (actually the 'keys') pertaining to each auditor who is entitled to decrypt the session – fake data is used for non-entitled auditors. The attributes of the table have the following format and meaning:

- **session_id**: this attribute is the foreign key of the table, therefore it's similar in format to its homonymic field in the table "Logs";
- **auditor_id**: this field represents a foreign key for the table, therefore it's similar in format to its homonymic field in the table "Auditors";
- **session_start**: as for the previous two fields, this attribute is a foreign key for the table, consequently it's similar in format to the field "session_start" (to whom it refers) in the table "Logs";
- **encrypted_key**: this field contains the data used by the Auditor to decrypt – when entitled to – a specific file relative to a session. Auditors who are not entitled to access the record are assigned fake data, as discussed in paragraph 3.3. Notice that the secrets pertaining to each auditor are encrypted with his public key before being archived in the DBMS;

Notice, lastly, that this is the only table where records are not signed with the private key assigned by the Log Manager. The reason of that difference is that the information inserted in this table is used to compute the hash chain whose rings, and whose related signatures, are inserted in the table "Logs" previously described. Consequently, a slight modification to this table would be detected simply by checking the integrity and authenticity of the chain itself.

5.8 System administration tools

Always focusing on the implementation, this section presents some system administration and configuration tools and their features. The system management requires, in fact, some tools which can be used by the Log Manager administrator to insert or edit – through the use of the DBcomm module – data concerning auditors and groups.

Notice that these tools must necessarily reside on the same computer where the Log Manager is installed. Furthermore, it's essential that the administrator may not be able to work on data concerning auditors and group without these tools. The reason is that by editing the auditors and groups tables an administrator could grant access to auditors who are not entitled to. What's worst, all of this could be done without being logged or controlled.

By forcing the administrators to use the provided tools, we are able to detect (a posteriori) and consequently eventually prevent unfair behaviors through the use of the system Assolo itself.

The tools designed to administer the Log Manger can be subdivided into 4 main areas, which are now described in details:

5.8.1 Auditors anonymous access management

It's necessary to implement some tools the administrators will be entitled to use to grant or deny access to the single auditors or groups.

To grant the access to an auditor the administrator must fill the table "Auditors" in the database with his personal data. More precisely, the tool will have to require as input all the data pertaining to the auditor (personal data, public key certificate, etc…), to assign him a 16 bit unambiguous identification number whose first bit must be 0, and finally to send all the data received to the DBcomm. The latter will, in turn, take care of signing the data with its private key and then sending them to the DBMS.

Notice that when access is granted to an auditor, the field "activation_date" of the table "Auditors" contains the time in which the auditor has been added, while the field "deactivation_date" will be set to NULL.

Data pertaining to users in the table "Auditors" must never be deleted, because it's necessary to maintain an historical list of auditors which have been entitled to access data, even if only for a short period of time. Consequently, when the administrator must revoke the access to an auditor, the revocation data must be inserted in the field "deactivation_date" related to that auditor. From this date on, the Log Manager will not generate secrets pertaining to the revoked auditor.

Notice that when an auditor is revoke, it must be removed also from the groups he belongs to, taking care not to break – with the removal – the minimum number of group elements necessary to build the secret information. This process is described in details in the following paragraphs, where group access management tools will be presented.

Notice, furthermore, that when an auditor is revoked access privileges, he will not be able to be reinserted with the same unambiguous identification number. The reason is that there is no record removal in the historical list of auditors which have been entitled (and of periods in which they were entitled). In case of auditor reinsertion, a new record in the table "Auditors" must be inserted, it would be wrong modifying the record previously belonging to him, editing the attributes values "activation_date" and "deactivation_date".

Also, when a new record pertaining to an auditor is added, the DBcomm must sign it. Further observation: only the personal data and the attribute "activation_date" are editable.

5.9 Group access management

As already stated in chapter 5.8 it's necessary to implement tools userful to add/remove groups from the system. In this chapter we will describe those tools.

In order to insert a new group into the system, it's necessary that all the auditors belonging to that group are already part of the system database. This is motivated by the fact that even if an auditor of the group is not entitled to access data by himself, he will be treated exactly as the others, hence preserving the property of elusion described in paragraph 3.4.

The tool used to insert the new group into the system will, firstly, have to take as input the number of elements belonging to the group and the minimum number of elements who have to collaborate to obtain the shared secret. By means of those two values, the Log Manger can create the exact number of sub-secrets pertaining to each auditor. Those sub-secrets must be generate in such a way that a sub-set of those secrets may lead to the key with whom the data were encrypted.

Subsequently, the tool will have to require as input all the identification numbers of the auditors belonging to the group. He will then check that they are all active and put them into the table "Groups_aud" accordingly.

As already stated for the auditors, when new records are inserted in the table "Groups" and "Groups_aud", it's necessary to sign them with the Log Manager's private key and to set the fields "activation_date" of the two tables respectively to the date in which the group was inserted and to the date in which the user has been inserted into the group. Again, the field "activation_date" will be set to NULL.

Also in this case, data pertaining to a group cannot be delete from the DBMS, therefore to revoke a group it's necessary to set the field "deactivation_date" to the date starting from which the group has no access privileges to data. At the same time, it's necessary to set the homonymic field contained in the table "Groups_aud" and pertaining to auditors belonging to that group, to the same value.

Notice that also in this case, when a record is modified must be anew signed; to be precise, the only attributes which can be modified are "group_info" (which contains generic information concerning a group) and "minimum_auditors" (the minimum number of auditors necessary to unfold/build a secret).

It must be observed that, when the field "minimum_auditors" is modified, it's necessary to check that the new value is lower or equal to the value of "total_auditors". Furthermore it's necessary to maintain an historical list of the characteristics of each group during its life (i.e. its activation period). For that reason, when the field “minimum_auditors” is modified, the preceding values are stored in the table “Groups_history”, setting the field “modification_date” accordingly.

The value of the field “total_auditors” is not immediately editable; we will show in the following chapter the way how it’s changed when an auditor is added or removed from a group.

5.10 Group members management

The system must include a tool to modify the composition of the single groups: it must be possible to add an auditor to a group or remove him from one of the groups he belongs to.

When an auditor has to be added to an existing group, his data must be already present in the table “Auditors” and the auditor itself must be active. The tool developed to make such modifications takes as input the identification of the group the auditor is joining to and the identification of the auditor itself.

Data concerning this association will be inserted into the table “Groups_aud”, and the value in the field “total_auditors” of the table “Groups” will be increased by one. Notice that, as stated before, it’s necessary to maintain an historical list of each group composition, consequently the previous data concerning the group members are inserted in the table “Groups_history”, setting accordingly the field “modification_date”.

When and auditor must be removed from a group, it’s necessary to fill the field “deactivation_date” of the table “Groups_aud” accordingly, in order to maintain an historical list of the composition of the groups. Similarly to the addition of a new auditor in the group, it’s now necessary to modify the field “total_auditors” pertaining to the group the auditor belong to in the table “Groups”: in this case, this field must be decreased by one.

Notice that, before allowing the removal operation of an auditor from a group, the tool must make sure that the new value of the field “total_auditors” is greater or equal than the value of “minimum_auditors”; on the contrary, the tool will have to prevent the auditor from being revoked. If this should happen, the administrator must simply decrease the value of the field “minimum_auditors” by one and then launch the modification of the group composition.

Similarly to the insertion of a new auditor, when deleting an auditor from a group (and therefore modifying the field “total_auditors” of the table) it’s necessary to store in the table “Groups_history” the previous value pertaining to the group.

5.11 Data integrity and authenticity

The tool described in this chapter is the one used to check the integrity and authenticity of data contained in the database tables.

The administrator must be able to check whether the contents of the tables “Auditors”, “Groups”, “Groups_aud” and “Groups_history” is genuine. In order to do se, the system must check whether all the records contained in those tables are correctly signed. The signature must be verified against the Log Manger’s public key, checking that the signatures are authentic. Of course, the administrator must be able to carry on those tests both on all the tables and or one single table.

In order to check the authenticity and integrity of data contained in the tables “Logs” and “Logs_aud”, on the other hand, the hash chain (stored in the field “hash_chain” of the table “Logs”) must be verified. In particular, two kind of checks must be carried on: the first, less detailed, is the check on data contained in the tables of the DBMS; the second, more detailed, is the check on the single files.

The less detailed check is carried on by re-computing the hash chain and verifying that each ring is authentic. More precisely, starting from the first ring of the chain, it’s necessary to compute the next rings in the same way as it’s done during the storing phase, and then, each time a new ring is computed, the system must check whether the related signature – stored in the field “signature” of the table “Logs”, is authentic.

The more detailed check, on the other hand, affects also the files containing data related to each session. More precisely, this check can be carried on by executing the hash chain verification afore described, and then by verifying whether the hashes contained in the field “encrypted_data_hash” of the table “Logs” are the same as the hashes of the files stored in the path stated in the field “path” of the same table. This kind of check is of course much more exhaustive, but it’s also more expensive in terms of time an resources occupied.

Notice that only the auditors must be entitled to check data integrity and authenticity. As auditors may access the data contained in the DBMS also from remote workstations, the verification procedure must be invoked and executed locally to the Log Manager. The reason is that the movement of all the data contained in the database on the auditor’s workstation might require too much time, considering that the integrity and authenticity verification involves all the data and not only a small part.

5.12 DBMS access management

As introduced before it's necessary, in order to raise the security level of the whole system, to create restrictive access policy to the DBMS.

In particular, it turned out to be necessary to introduce different DBMS users, each one with different privileges concerning his specific task. Three distinct users have hence been created:

- **u_logmanager**: it's the user the Log Manager makes use of to access the DBMS;
- **u_tool**: it's the user the several tools described in the previous sections make use of in order to manage the data contained in the database;
- **u_auditor**: it's the user the auditors make use of in order to retrieve data from the database;

The first limitation enforced to these users is based on their location. We thought it was more safe to allow the users u_logmanager and u_tool to have access to the DBMS only if their IP address is the same as the computer the Log Manager is running on (notice that the access to the DBMS is carried out through a TCP/IP connection). The user u_auditor, on the other hand, can be used to access the DBMS from any workstation, because auditors are allowed to access the system from remote locations.

For security reasons, these three users are granted with the sole and only privileges they may need to engage correctly the archiving and visualization operations.

In particular, the user utilized by the auditors during the verification phase must have only the capability of reading the database tables. Auditors must, in fact, only be able to retrieve, by means of appropriate queries, the data pertaining to the groups composition, the auditors and data archived in the tables "Logs" and "Logs_aud". At the same time, auditors must not be able to edit, delete and add records to the DBMS.

The user u_logmanager, on the other hand, must be able to insert data in the tables "Logs" and "Logs_aud", other than reading the data pertaining to groups and auditors. In this way, the Log Manager can generate accordingly the secrets pertaining to each auditor. Notice that this user must not, nevertheless, be able to edit or delete records from the table "Logs" and "Logs_aud", because this could lead to the compromising of the hash chain.

Finally, the user who can be utilized for system management must be able to add to the DBMS data concerning auditors and groups, conversely he must not be entitled to access (read or write) data archived by the Log Manager. Notice that the management tools can be used also to modify some fields of the tables "Auditors", "Groups" and "Groups_aud"; for that reason, we decided to provide the user u_tool with the edit privileges only on those fields. Further observation: this user must not be entitled to delete data from any of the DBMS tables, not even those containing the data concerning groups and auditors. The removal of a user or a group, as described in the previous sections, does not cause the removal of data from the database, but the modification of the field "deactivation_date" of the record pertaining to the auditor or the group at issue.

MySQL – the DBMS adopted for this implementation – consents data management with granularity as subtle as the single table attribute. For each attribute we can specify, in relation with each user, any combination of the four available types of performable operations, that are:

- SELECT: used to read data from the database;
- INSERT: used to insert new data into the database;
- UPDATE: used to edit data already present in the database;
- DELETE: used to delete data from the database;

In the following figures there is a summarization of the privileges of the three users described above on the single tables of the DBMS:

Table "Auditors"			
Attribute	**u_auditor**	**u_logmanager**	**u_tool**
auditor_id	select	select	select, insert
personal_data	select	select	select, insert, update
activation_date	select	select	select, insert
deactivation_date	select	select	select, insert, update
public_key	select	select	select, insert
signature	select	select	select, insert, update
pubkey_certificate	select	select	select, insert, update

Table "Groups"			
Attribute	**u_auditor**	**u_logmanager**	**u_tool**
group_id	select	select	select, insert

group_info	select	select	select, insert, update
total_auditors	select	select	select, insert, update
minimum_auditors	select	select	select, insert, update
activation_date	select	select	select, insert
deactivation_date	select	select	select, insert, update
signature	select	select	select, insert, update
pubkey_certificate	select	select	select, insert, update

Table “Groups_aud”			
Attribute	**u_auditor**	**u_logmanager**	**u_tool**
group_id	select	select	select, insert
auditor_id	select	select	select, insert
activation_date	select	select	select, insert
deactivation_date	select	select	select, insert, update
signature	select	select	select, insert, update
pubkey_certificate	select	select	select, insert, update

Table “Groups_history”			
Attribute	**u_auditor**	**u_logmanager**	**u_tool**
group_id	select	/	select, insert
modification_date	select	/	select, insert
total_auditors	select	/	select, insert
minimum_auditors	select	/	select, insert
signature	select	/	select, insert
pubkey_certificate	select	/	select, insert

Table "Logs"			
Attribute	**u_auditor**	**u_logmanager**	**u_tool**
session_id	select	select, insert	/
user_pseudonym	select	select, insert	/
administered_server	select	select, insert	/
session_start	select	select, insert	/
session_end	select	select, insert	/
session_termination	select	select, insert	/
timestamp	select	select, insert	/
path	select	select, insert	/
encrypted_data_hash	select	select, insert	/
hash_chain	select	select, insert	/
signature	select	select, insert	/
pubkey_certificate	select	select, insert	/

Table "Logs_aud"			
Attribute	**u_auditor**	**u_logmanager**	**u_tool**
session_id	select	select, insert	/
auditor_id	select	select, insert	/
session_start	select	select, insert	/
encrypted_key	select	select, insert	/

Figure 5-7: Privileges of the users on the single tables

Chapter 6
Conclusion and future work

In this chapter we present a short discussion about what has been done so far and what can still be done starting from the present work. Furthermore, some reasoning on the ethical issues related to the field of *irrefutable administration* is put forward.

6.1 Summing up the results

In this thesis we've presented the newly-born concept of *irrefutable administration* and it's links to the background technologies involved in the process. The state of the art found in the first part introduces us into the world of cryptology and related concepts: hash functions, symmetrical/asymmetrical encryption, public key cryptography, secret sharing and group signatures.

Starting from general concepts, going up to group signatures, we've faced the subject of restricted data access with anonymity, privacy and authentication moving our steps through concepts like encryption/exclusion/elusion/group-access.

We've devised two new methods for group signatures and a complete framework concerning secure logging for irrefutable administration. The two group signatures schemes are based on different technologies and theoretical principles: the first (paragraph 4.3) is based on standard signatures arranged in a framework which allows signatures to be made by singles on behalf of a group and allows also the group manager to revoke group members. The same goes for the second scheme (paragraph 4.4) which is, on the other hand, based on completely different theoretical basis, i.e. a concept known as *one-way accumulators*. The *irrefutable administration* scheme is, indeed, a complete framework for building a system which archives logs in a secure and anonymous manner, allowing set of users to have access to those logs under some precise constraints. The system here conceptually described has been implemented, in collaboration with the Department of Computer Science, University of Turin, into a working environment called *Assolo*, described in details in Chapter 5.

6.2 Irrefutable administration and its ethical issues

We are conscious that the field we've so far called *irrefutable administration* offers a lot of hints not only on what can and what can't be done or improved, but also on what is ethically correct and what is not. A lot of fuss is coming out nowadays on the problem of employees monitoring, mainly about email messages and network activities. According to [RJ05] last year ZDNet reported on a Proofpoint-sponsored Forrester survey stating that 44 percent of companies read outgoing email. Currently, most of those companies employ human beings to read that email, but automated processes to scan content aren't far behind. To show how fast the trend is spreading, a newer survey from Forrester Consulting recently states that 63 percent plan to monitor outgoing mail. The survey also states that as Instant Messaging becomes more prevalent, those companies plan to monitor IM traffic as well. Employers are spying on their employees because they don't trust them. And worse, they're not spying only on employees that they suspect of breaking trust and leaking information — they're spying on everyone, because technology lets them do so.

Irrefutable administration goes a step further, because it's more specific but also more ethical, in our opinion and in the framework we devised. What we mean is, there are contexts in which an appropriate monitoring and archiving – under the constraints of encryption and group access which guarantee non disclosure of critical information – is advisable both for the company and for the employees. Resuming the example cited in Chapter 1, in industrial environments some jobs are left in outsourcing to external companies; the operations performed by the external personnel should be controlled in some way, and at the same time, the privacy of the workers must be guaranteed, with the ability to verify and link, in case of necessity, the operations performed to the person who made them. The "case of necessity" might be a court action against the employee but also – and here lays the counterpart of the whole idea – a defense of the employee or even a court action against the company. Companies have their right to be protected, but the same right is owned by the employees. Irrefutable administration must be conceived in that spirit, of ambivalent protection of both sides.

6.3 Future work

The solutions devised in this thesis are complete but, in some cases, some improvements can be conceived in order to fix some drawbacks keeping, at the same time, the benefits of the methods. This paragraph contains a simple and short summary of the point which have been deeply analyzed in the dissertation.

The solutions presented for group signatures still suffers from some minor drawbacks, such as the need of an intermediate entity in the first method and the need of a sort of CLR (*Certificate Revocation List*) of the second method. Some investigation can be made on those subjects, in order to provide a better group signatures scheme, keeping in mind the tradeoffs between performance, availability, key distribution, key dimension, etc… The irrefutable administration subject can be improved working on the intermediate entity as well, or analyzing more in details the archiving/logging function other than the auditing and data-retrieving side.

Chapter 7
Bibliography

[AAB98] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, P. G. Neumann, R. L. Rivest, J. I. Schiller, B. Schneier. The Risks Of Key Recovery, Key Escrow, And Trusted Third-Party Encryption. http://www.cdt.org/crypto/risks98

[ACJT00] G. Ateniese, J. Camenisch, M. Joye, G. Tsudik. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In L. Bellare, editor, *Advances in Cryptology-Crypto '2000*, volume 1880 of LNCS, pages 255-270. Springer-Verlag, 2000

[AST02] G. Ateniese, D. Song and G. Tsudik. Quasi-Efficient Revocation of Group Signatures. In *Financial Cryptography 2002*, Southampton, Bermuda, March 11-14, 2002

[AT99] G. Ateniese and G. Tsudik. Some Open Issues and New Directions in Group Signature. In *Financial Cryptography '99*, 1999

[BCC01] F. Bergadano, D. Cavagnino, B. Crispo. Chained Stream Authentication. *Proc. of Selected Areas in Cryptography 2000*, D. R. Stinson and S. Tavares, eds., LNCS, no. 2012, Springer-Verlag, pp. 144-157

[BCD+04] F. Bergadano, D. Cavagnino, P. Dal Checco, A. Nesta, M. Miraglia, P. L. Zaccone. Secure Logging for Irrefutable Administration. Paper submitted.

[BCD+05] F. Bergadano, D. Cavagnino, P. Dal Checco, P. L. Zaccone, M. Leone, E. Caprella. Off-line Group Signatures. Paper waiting for submission (due to NDA constraints)

[BCE02] F. Bergadano, D. Cavagnino, L. Egidi. Partially sighted signatures on large documents. *Proc. of Int.l Network Conference 2002*, Sherwell Conference Centre, University of Plymouth, UK, pp. 373-380

[BCN01] F. Bergadano, D. Cavagnino, P. A. Nesta. Certification of Web Access Statistics. *Proc. of e-2001, E-work and E-commerce*, Vol. 1, IOS Press, October 2001, pp. 326-332

[BDP97] A. Bosselaers, H. Dobbertin, B. Preneel. The RIPEMD-160 cryptographic hash function. *Dr. Dobb's Journal*, January 1997

[BLA79] G. R. Blakley. Safeguarding cryptographic keys. *Proc. of AFIPS, 1979 NCC*, Vol. 48, Arlington, Va., June 1979, pp. 313-317

[BM94] J. Benaloh, M. de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In: *Advances in Cryptography – Eurocrypt '93*, LNCS 765, pages 274-285. Springer-Verlag, Berlin, 1994

[BP97] N. Baric and B. Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. *Eurocrypt '97, LNCS 1233*, Springer-Verlag, Berlin 1997

[BRS01] E. Bresson, J. Stern. Efficient Revocation in Group Signatures. In K. Kim, editor, *Public Key Cryptography-PKC2001*, volume 1992 of LNCS, pages 190-206. Springer-Verlag, 2001

[BY97] M. Bellare, B. S. Yee. Forward integrity for secure audit logs. *Tech. report*, UC at San Diego, Dept. of Computer Science and Engineering, November 1997

[CA98] J. Camenish. Group signature schemes and payment systems based on the discrete logarithm problem. *PhD thesis*, vol. 2 of *ETH Series in Information Security and Cryptography*, Hartung-Gorre Verlag, Konstanz, 1998

[CDG+03] Joris Claessens, Claudia Díaz, Caroline Goemans, Bart Preneel, Joos Vandewalle, Jos Dumortier. Revocable anonymous access to the Internet?. Internet Research: Electronic Networking Applications and Policy. August 2003

[CE04] Federica Cesano. Visualizzazione ed auditing dei dati derivanti dall'amministrazione di un sistema informatico. Degree Thesis, 2004

[CHVH91] D. Chaum and E. van Heyst. Group Signatures. In D.W. Davies, editor, *Eurocrypt '91*, volume 547 of LNCS, pages 257–265. Springer-Verlag, 1992

[CL02] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In: *Crypto 2002*, LNCS 2442, pp. 61-76. Springer-Verlag, 2002

[CM98a] J. Camenisch and M. Michels. A group signature scheme with improved efficiency. In *Advances in Cryptology | ASIACRYPT'98*, vol. 1514 of LNCS, pp. 160{174, Springer-Verlag, 1998

[CM98b] J. Camenisch and M. Michels. A group signature scheme based on an RSA-variant. *Technical Report* RS-98-27. BRICS, University of Aarhus, November 1998. Primary version of this paper appeared at *ASIACRYPT'98*

[CO96] D. Coppersmith. Finding a small root of a bivariatre interger equation; factoring with high bits known. In *Advances in Cryptology | EUROCRYPT '96*, volume 1070 of LNCS, pages 178{189. Springer Verlag, 1996

[CPH02] C. N. Chong, Z. Peng, P. H. Hartel. Secure Audit Logging with Tamper-Resistant Hardware. 18th IFIP TC11 *International Conference on Information Security (IFIPSEC)*, Security and Privacy in the Age of Uncertainty, 2003, pp. 73-84

[CS97] J. Camenisch and M.Stadler. Efficient Group Signatures Schemes for Large Groups. In B. Kaliski, editor, *Crypto '97*, volume 1294 of LNCS, pages 410–424. Springer-Verlag, 1997

[DES93] Yvo Desmedt. Threshold cryptography. *Auscrypt '92*, LNCS 718, Springer-Verlag, Berlin 1993, 3–14

[DH76] W. Diffie, M. E. Hellman. New direction in cryptography. *IEEE Trans. On Inform. Theory*, Vol. 22, pp 644-654, 1976

[DR00] J. Daemen, V. Rijmen. The Block Cipher Rijndael. Smart Card Research and Applications, J.-J. Quisquater and B. Schneier, eds., Lecture Notes in Computer Science, no. 1820, Springer-Verlag, 2000, pp. 288-296

[DTX04] X. Ding, G. Tsudik and S. Xu. Leak-Free Mediated Group Signatures. *IEEE ICDCS'04*, March 2004

[KPW96] Seungjoo Kim, Sungjun Park and Dongho Won. Convertible group signatures. *Advanced in Cryptology - AsiaCrypt'96*, Springer-Verlag, LNCS 1163, Kyongju, Korea, November 3-7 1996, pp.311-321

[LAM81] L. Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24 (1981), pp. 770-772

[MI04] Michele Miraglia. Registrazione sicura, anonima e non disconoscibile dei dati derivanti dall'amministrazione di un sistema informatico. Degree thesis, 2004

[ML05] Manuel Leone, Comments to the group signatures scheme 'Group signatures with immediate verification, TILab internal document, 2005

[MOV96] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996

[NIST94] National Institute of Standards and Technologies, NIST FIPS PUB 186, "Digital Signature Standard", U.S. Department of Commerce, May 1994

[NIST95] National Institute of Standards and Technology, NIST FIPS PUB 180-1, "Secure Hash Standard", U.S. Department of Commerce, Apr. 1995

[NIST99] National Institute of Standards and Technologies, NIST FIPS PUB 46-3, "Data Encryption Standard (DES)", U.S. Department of Commerce, October 1999

[NIST01] National Institute of Standards and Technologies, NIST FIPS PUB 197, "Advanced Encryption Standard (AES)", U.S. Department of Commerce, Nov. 2001

[NIST02] National Institute of Standards and Technologies, NIST FIPS PUB 180-2 “Secure Hash Standard”, U.S. Department of Commerce, Aug. 2002

[PER01] A. Perrig. The BiBa one-time signature and broadcast authentication protocol. In Eighth ACM Conference on Computer and Communication Security, pages 28-27. ACM, November 5-8 2001

[PGP05] PGP Corporation – Home Page. http://www.pgp.com/

[REY02] L. Reyzin, N. Reyzin. Better than BiBa: Short One-time Signatures with Fast Signing and Verifying. ACSIP 2002

[RJ05] Russell Jones. Monitoring Technologies Put Developers in an Ethical Hotseat. *http://www.devx.com/opinion/Article/28657*

[RSA78] R. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, Communications of the ACM, 21 (1978), pp. 120-126

[RUF95] M. Ruffin, “A survey of logging uses”, Tech. Rep., Dept. of Computer Science, University of Glasgow, Glasgow, Scotland, February 1995

[SCH94] B. Schneier, “Description of a new variable-length key, 64-bit block cipher (Blowfish)”, Fast Software Encryption, Cambridge Security Workshop Proceedings, Lecture Notes in Computer Science, no. 809, Springer-Verlag, 1994, pp. 191-204

[SHA79] A. Shamir, “How to share a secret”, Communications of the ACM, 22 (1979), pp. 612-613

[SK98] B. Schneier, J. Kelsey, “Cryptographic support for secure logs on untrusted machines”, in The 7th USENIX Security Symposium Proceedings, pp. 53-62, USENIX Press, January 1998

[SK99a] B. Schneier, J. Kelsey, “Secure Audit Logs to Support Computer Forensics”, ACM Transaction on Information and System Security, Vol. 2, Issue 2 (May 1999)

[SK99b] J. Kelsey, B. Schneier, “Minimizing bandwidth for remote access to cryptographically protected audit logs”, in Web proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID'99), 1999

[SK99c] B. Schneier, J. Kelsey, "Event Auditing System", US Patent #5,978,475, Nov. 2, 1999

[TX03] Gene Tsudik and Shouhuai Xu. Accumulating Composites and Improved Group Signing. In: ASIACRYPT 2003, LNCS 2894, pp. 269–286. Springer-Verlag, 2003. Primary version available at http://eprint.iacr.org/2003/112/

[WBD04] B. R. Waters, D. Balfanz, G. Durfee and D. K. Smetters, "Building an Encrypted and Searchable Audit Log", The 11th Annual Network and Distributed System Security Symposium (NDSS 2004), February 2004