Dipartimento di Informatica e
Scienze dell'Informazione

Secure and Efficient Group Communication in Wide and Local Area Networks

by

Giuseppe Ateniese

Theses Series DISI-TH-1999-XX

DISI, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy http://www.disi.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica e
Scienze dell'Informazione

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

Secure and Efficient Group Communication in Wide and Local Area Networks

by

Giuseppe Ateniese

December, 1999

Dottorato di Ricerca in Informatica
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova

DISI, Univ. di Genova
via Dodecaneso 35
I-16146 Genova, Italy
http://www.disi.unige.it/

Ph.D. Thesis in Computer Science

Submitted by Giuseppe Ateniese
DISI, Univ. di Genova
ateniese@disi.unige.it

Date of submission: October 1999

Title: Secure and Efficient Group Communication in Wide and Local Area Networks

Advisor: Prof. Giovanni Chiola
Department of Computer Science (DISI)
University of Genoa, Italy
chiola@disi.unige.it

Ext. Reviewers: Prof. Yair Amir
Department of Computer Science
The Johns Hopkins University, USA
yairamir@cs.jhu.edu

Prof. Gene Tsudik
Department of Computer Science
University of California (Irvine), USA
gts@isi.edu

Abstract

Group communication occurs in many different settings: from low-level network multicasting to conferencing and other groupware applications. Regardless of the environment, security services are necessary to provide communication privacy and integrity. These are not possible without secure and efficient key distribution, authentication and other security mechanisms.

Given the openness of today's networks, communication among group members must be secure and, at the same time, efficient. While peer-to-peer security is a well-developed field, secure group communication remains relatively unexplored. The prime motivation of this dissertation is the lack of a general-purpose group security solution that satisfies the requirements of modern group communication.

Secure group communication is not a simple extension of secure two-party communication. Two-party communication can be viewed as a discrete phenomenon: it starts, lasts for a while and ends. Instead, group communication starts, the group mutates (members leave and join) and there might not be well-defined end. All the existing protocols start from scratch in case of membership changes. This dissertation investigates more efficient alternatives as well as new definitions and services in group setting.

The next step is to analyze a general-purpose toolkit and API for key agreement and related security services. Finally, the design and implementation of secure protocols over a multicast communication layer (SPREAD) is described. A multicast communication layer facilitates the development of fault tolerant distributed applications in a network of machines. It guarantees all-or-none delivery semantics, and handles message losses and transient network failures transparently. This dissertation discusses some important integration issues, reports on the implementation experience and provides experimental results.

To my wife Carmen for her Love

Success is to be measured not so much by the position that one has reached in life as by the obstacles which he has overcome while trying to succeed.

(Booker T. Washington)

Acknowledgements

First of all, I would like to thank my advisor Giovanni Chiola for his support and encouragement. His endless patience and insightful feedbacks allowed me to complete my research activity.

This dissertation could never have come to light without the support of many people. My sincerest thanks go to: Yair Amir, Jan Camenisch, Damian Hasse, Marc Joye, Yongdae Kim, Cristina Nita-Rotaru, Theo Schlossnagle, John Schultz, Jonathan Stanton, Michael Steiner, and Gene Tsudik.

I am particularly and deeply indebted to Jan Camenisch, Marc Joye and Gene Tsudik for their contributions toward this work.

I thank Alfredo De Santis at the University of Salerno for introducing me to cryptography.

I thank the entire Computer Science Department at the University of Genoa for supporting me and giving me the opportunity to be enrolled in their first-class PhD program.

I thank the Information Sciences Institute at the University of Southern California for giving me the opportunity to work with Gene Tsudik within the DARPA project CLIQUES (`http://www.isi.edu/ gts/CLIQUES/`).

I thank the Network Security group at IBM Zurich Research Lab where I spent a great time learning and discussing many aspects of computer security and cryptography. In particular I would like to thank: Christian Cashin, Luke O'Connor, Victor Shoup, Michael Steiner, Els Van Herreweghen, and Michael Waidner.

Finally, I would like to make two special thanks. The first goes to Gene Tsudik. Without his guidance, insights and teachings, I could never have concretized the results contained in this dissertation. Gene is a great scientist as well as a great person and an invaluable friend.

The second goes to my wife Carmen, who is constantly and patiently teaching me what is really important in life.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Just as many-to-many peer group communication tends to be much more complex than two-party communication, security in a multi-party setting is much harder to define, specify and achieve, than in a two-party case.

Communication channel specifics have relatively little impact on the ability of **two** parties to communicate securely. An unreliable communication channel may drop or corrupt data but, at worst, it can only prevent communication. Well-known cryptographic methods can be utilized to assure data privacy, data integrity, source authentication and other properties. Moreover, techniques exist for hiding the communication patterns, thus preventing hostile traffic analysis. As a result, the adversary's choice is reduced to a binary one: either allow communication or prevent it altogether.

In contrast, secure group communication is very much dependent on the composition of the group. A group, unlike a pair of end-points, mutates over time. If we collapse all pair-wise channels within a group into a single group channel, its state cannot be expressed as a binary value. Fluctuations in the group channel state cause and are caused by members joining and leaving the group. To achieve the highest level of security, every state fluctuation must be accompanied by a corresponding adjustment to group security parameters. Of these, the most apparent is the group shared keying material or group secret.

This leads to a classical case of a *chicken-and-egg* problem: security events must immediately follow group state change events yet the latter are not themselves secure. However, we argue that certain group communication events are fundamentally impossible to secure. Notably, these include all kinds of fault-like events leading to group partitions and involuntary member disconnects. As a concrete example, consider the situation where network faults and individual node disconnects constantly perturb group membership. In this setting, trying to differentiate between two possible causes: 1) a clever adversary and 2) a truly faulty network, is impossible. Other events, such as members joining a group (whether as singletons or *en masse*) can, indeed, be made secure.

Finally, there is the all-important matter of trust. Trust, a very vague notion in a two-party case, becomes even vaguer in a group context. As group membership changes, trust among group members may change over time.

1.1 Security Goals

Our main security goal is both natural and fairly standard: to achieve authentic and private communication within a group. Although this requirement can be expressed in any secure group setting–in a dynamic peer group context–it leads to a number of interesting corollaries. The second security goal is to provide authentic and private communication between a secure group (i.e., its members) and other entities (non-members). Finally, the third goal is to obtain strong authentication and non-repudiation of individual group members both within and outside a group.[1]

So-called *conventional* cryptography requires two (or more) parties to share a common secret in order to communicate securely. In contrast, public key cryptography [49] allows two parties who do not share a common secret to communicate securely. Owing to its computational cost (orders of magnitude greater than that of conventional encryption) public-key encryption can be used to secure communication between two parties only when a very small amount of data is involved. In practice, public-key encryption is used only as a means to distribute (or agree on) a common secret key which is subsequently used with conventional cryptography to provide authenticity and privacy of bulk data.

To obtain security in a peer group setting, we can construct an obvious extension to any two-party security mechanism (such as SSL or SSH) by establishing N^2 (where N is the group size) pair-wise secure channels among all group members. Each pair-wise channel would then be associated with a unique secret key known only to the two endpoints. In order to send private data to the group, the sender would have to encrypt it $(N-1)$ times (once for every recipient) producing as many copies which would then be sent individually to each recipient or conglomerated into a single message and broadcasted to the entire group. It is easy to see that this approach involves horrendous overhead both in terms of computation and bandwidth consumed.[2]

[1] That is, a secure group can be viewed as a microcosm of sorts where authentication and non-repudiation of individuals is at the granularity of members, not permanent identities.

[2] We note that the only exception is its use in secure multi-destination electronic mail, such as PEM [64], where recipient “groups” are constructed on a per-message basis.

1.2 Group Key Management

The above discussion essentially implies that–practically speaking–neither private nor authentic communication within a group can be achieved without some sort of a common group secret. Consequently, the starting point for security is a set of mechanisms for obtaining and maintaining such a secret. We refer to this collectively as the group key management problem.

Besides the fact that it forms the basis of all other security services, there are two other important reasons for our focus on group key management:

- The security of group key management itself is of paramount importance. Data privacy and authenticity (integrity) mechanisms are formed by defining a message format and selecting a basic underlying algorithm, i.e., a block cipher such as DES[83][3] for privacy and a keyed MAC[4] such as HMAC [69] for data integrity. Whereas, key management mechanisms involve, in addition, intricate *protocols*. An adversary aiming to attack a system is unlikely to waste efforts on attacking privacy and integrity mechanisms since key management presents a much more attractive target.
- The cost of group key management represents a "pure" form of security overhead. Compared with the cost of encryption (which can be done with almost no overhead if certain types of stream ciphers are used) or integrity (which can be obtained with certain Gbit-speed MACs), key management typically requires relatively heavy-weight arithmetic operations and additional communication among group members.

1.3 Group Security In Distributed System

Regardless of the environment, security services are necessary to provide communication privacy and integrity. These are not possible without secure and efficient key distribution, authentication and other security mechanisms.

It is possible to apply several security services to multicast communication, including authentication, confidentiality and integrity. In particular, the following general goals should be considered when designing a secure system:

- *Authentication.* It provides assurance of a group member's identity. For instance, it requires that a received message was actually sent by the claimed sender. Authentication, in multicast communications, is generally achieved via digital signatures.

[3] Data Encryption Standard.
[4] Message Authentication Code.

- *Access Control.* Access control mechanisms use the group member's identity as the basis on which to allow or deny a specific action, e.g., joining or sending data to a group. It can be achieved via *access control list*.

- *Integrity.* It provide assurance that multicast traffic cannot be altered during transmission without detection. It is generally achieved via keyed hash functions.

- *Confidentiality.* It prevents information disclosure to unauthorized parties (*data confidentiality*). A stronger requirement would ensure that origin, destination, and volume of data transmitted are not disclosed (*traffic confidentiality*). Confidentiality is usually achieved via encryption techniques.

- *Non-repudiation.* It provide assurance that the receiver (sender) can prove that the sender (receiver) did send (receive) the message. This requires that each message is digitally signed.

To implement security services required for a multicast session, it is important to specify parameters such as type of cryptographic algorithms and authentication mechanisms, key length, time after which key material must be refreshed, and other security related information. In a peer-to-peer session, these parameters can be negotiated between hosts. In group communication, session requirements are typically defined by an initiator, called *group controller*, and announced to the other group members.

1.3.1 Secret Key Agreement

Secure group communication is not a simple extension of secure two-party communication. Two-party communication can be viewed as a discrete phenomenon: it starts, lasts for a while and ends. Instead, group communication starts, the group mutates (members leave and join) and there might not be well-defined end. All the existing protocols start from scratch in case of membership changes. Chapter 2 investigates more efficient alternatives as well as new definitions and services in group setting. In particular, a new contributory key agreement protocol is developed that allows two or more parties to agree on a secret value. More formally, a contributory key agreement protocol is a key establishment technique whereby a shared secret key is derived by two (or more) parties as a function of fresh information equally contributed by each of these. A protocol of such type is said to provide group implicit key authentication if each party of the group is assured that no other party outside the group itself may compute a particular secret key. These features allow a group to share efficiently a key such that the resulting scheme is resistant to active attacks (i.e. where an attacker is able to modify, delay or inject messages).

The primary motivation for obtaining a group key is the ability to communicate securely and efficiently once a key is established. If all group members share a key, they can communicate using symmetric encryption. This is more efficient than schemes not requiring key establishment.

To satisfy the requirements of high-level security group communication we have developed a key agreement protocol providing:

- Perfect Forward Secrecy (PFS), i.e., compromising of a long-term key(s) cannot result in the compromise of past session keys. That is, the idea is that previous traffic is locked securely in the past.
- Resistance to known-key attacks, i.e. compromising of past session keys does not allow an active adversary to compromise future session keys or impersonate one of the protocol parties.
- A formal proof of security, i.e. ideally it should be possible to base all protocol security properties (including PFS and resistance to known-key attacks) on a single, well-studied, hard problem (hence, excluding any ad-hoc solution as well as conventional encryption).

1.3.2 Security Services: Group Signature

Once group members share the same key, they can generate signatures via appropriate cryptographic techniques (e.g., using MAC functions). The resulting signatures are completely anonymous and unlinkable, i.e., there is no way to determine the identity of the signer or establish whether multiple signatures were produced by the same group member. However, only few environments would allow this behavior, it is rather preferable to have the anonymity revoked by some trusted authority in case of disputes. This concept brings to *group signature schemes*, extensively treated in chapter 3. Group signatures have many practical applications and it turns out to be a very useful security service.

Shortly, a group signature scheme allows any member of a group to sign on behalf of the group. The resulting signatures are anonymous and unlinkable for everyone with the exception of a designated group manager who can reveal the identity of the actual signer in case of dispute. At the same time, no one (including the group manager) can misattribute a valid group signature. In chapter 3 we construct a very efficient group signature scheme with security properties based on a set of well-studied (and widely believed to be hard) computational problems.

1.3.3 Other New Services

There are many interesting group services that should be considered. In particular:

- Authentication to outsiders
- Intra-group authentication
- Non-repudiation of group membership
- Private communication within group
- Private communication between outsiders and groups

We note that all the previously mentioned group services can be achieved using a shared group key. For example, we can use a secret group key to derive a corresponding group public key which can be subsequently embedded in a group certificate. This would allow any group member to use digital signatures to authenticate itself (as group member) to both insiders and outsiders. The same group public key can be used by outsiders (including other groups) to establish shared keys with the entire group. Similarly, outsiders can use the group public key to communicate in secret with the entire group. See chapter 4 for details.

One novel and practical service that can be built using group signatures as a primitive is *multi-group signatures* [15]. Informally, a multi-group signature (MGS) is a set of regular group signatures generated by the same entity (signer) who is a member of multiple groups. For example, a user belonging to two groups A and B may have a reason to sign something as a member of both groups. Another interesting new group service that can be built on top of group signature schemes, is *sub-group signatures* [15]. Informally, a sub-group signature (SGS) is a group signature generated by a subset of group members. The goal of SGS is to demonstrate that a subset of a certain (unknown a priori, but verifiable) size has signed a given message. Chapter 4 presents sample realizations of these services.

1.3.4 Overview of CLIQUES and CLQ_API

CLIQUES [102, 103, 14] is a cryptographic protocol suite which provides authenticated contributory group key management and other security services. CLIQUES protocols guarantee key independence, key confirmation, perfect forward secrecy and resistance to known key attacks. (See chapter 2 for detailed definitions of these attacks.) In short, CLIQUES guarantees that group keys cannot be obtained by either active or passive attackers. Moreover, past group members (alone or in collusion) cannot obtain group keys subsequent to leaving the group and, similarly, current group members cannot obtain group keys used before they joined the group. CLIQUES is based on group extensions of the well-known Diffie-Hellman [49] key exchange technique, as treated in chapter 2.

CLIQUES defines a special role for a group controller, the last member to join a group. This role floats as the group membership changes. A controller is charged with initiating key adjustments following membership changes. Other than that, it has no special security tasks or privileges.

CLQ_API [10] is a group key agreement API built on top of the CLIQUES protocol suite. Its main purpose is to implement the cryptographic primitives of CLIQUES. The underlying communication system is assumed to deal with the group communication and network events such as partitions, failures and other abnormalities. CLQ_API is small and concise containing only eight function calls, all described in chapter 5.

CLIQUES guarantees two system invariants:

1. All group members always agree on the identity of the current group controller. At any point in time, the controller is the newest (most recently joined) group member.
2. A group secret key is always contributed to equally by each and every group member. At any point in time, a group secret is a function of all members' private shares. Only current group members have access to the group secret.

CLIQUES supports the following group key agreement operations:

- Join: a new member is added to the group.
- Merge: one or more members are added to the group.
- Leave: one or more members are removed from (or leave) the group.
- Key Refresh: generates a new group secret.

All of these result in the same outcome, i.e., the group secret is changed such that: all members obtain the same authenticated key.

1.3.5 Integration and Experiments

Finally, the design and implementation of secure protocols over a multicast communication layer (SPREAD) is described in chapter 6. Our approach utilizes distributed group key management developed by the CLIQUES project. We enhance it to handle processor and network faults (under a fail-stop or crash-and-recover model) and asynchronous membership events (such as cascading joins, leaves, merges and network partitions). Our approach leverages the strong properties provided by the SPREAD group communication system, such as message ordering, clean failure semantics and a membership service. The result of this work is a secure group communications layer and an API which provide the application programmer both standard group services as well as flexible security services.

Chapter 2

Authenticated Group Key Agreement

Many modern computing environments involve dynamic peer groups. Distributed simulation, multi-user games, conferencing applications and replicated servers are just a few examples. Given the openness of today's networks, communication among peers (group members) must be secure and, at the same time, efficient. This chapter studies the problem of authenticated key agreement in dynamic peer groups with the emphasis on efficient and provably secure key authentication, key confirmation and integrity. It begins by considering 2-party authenticated key agreement and extends the results to Group Diffie-Hellman key agreement. In the process, some new security properties (unique to groups) are encountered and discussed. The results described in this chapter have appeared in [13] and [14].

2.1 Introduction

This chapter is concerned with security services in the context of dynamic peer groups (DPGs). Such groups are common in many layers of the network protocol stack and many application areas of modern computing. Examples of DPGs include replicated servers (such as database, web, time), audio and video conferencing and, more generally, collaborative applications of all kinds. In contrast to large multicast groups, DPGs tend to be relatively small in size, on the order of a hundred members. (Larger groups are harder to control on a peer basis and are typically organized in a hierarchy of some sort.) DPGs typically assume a many-to-many communication pattern rather than one-to-many commonly found in larger, hierarchical groups.

The specific security requirements and needs of dynamic peer groups – in particular, key management – are still considered as open research challenges [99]. Recently, several key agreement protocols geared for DPGs were proposed in [102]. They were obtained by extending the well-known Diffie-Hellman key exchange method [49] to groups of n parties. These protocols perform what we refer as *initial key agreement* (IKA) within a group. Once a group is formed and the initial key is agreed upon, group members may leave (or be excluded) and new members may join. Moreover, entire groups may join and entire sub-groups may need to be excluded. Any membership change must cause a corresponding group key change in order to preserve *key independence*.[1] Since re-running full IKA for each membership change is expensive, other supporting protocols are necessary. The operations supported by these protocols are collectively called *auxiliary key agreement* (AKA). AKA protocols, also based on Diffie-Hellman extensions, have been developed in [103]. Both IKA and AKA protocols have been shown secure against passive adversaries. (The security is based on the polynomial indistinguishability of a Diffie-Hellman key from an arbitrary random value.)

This chapter leverages the results of [102, 103] to develop practical and secure *authenticated* key agreement protocols for DPGs. Also considered are other relevant security features such as key confirmation, key integrity and entity authentication. In doing so, we discover that the meaning of these and other familiar notions need to be redefined in a peer group setting.

Our goal is the development of a comprehensive protocol suite and a toolkit for secure communication in DPGs. Although the focus is on relatively small non-hierarchical peer groups, no specific communication paradigm (e.g., RPC, connection-oriented) is favored, and no assumptions are made about either the topology or technology of the underlying network.

2.2 Key Establishment Protocols

Key establishment protocols can be roughly classified in two categories: *key agreement* protocols [103] and (centralized) *key distribution* protocols based on some form of a trusted third party (TTP). Although, we focus on contributory key agreement, we briefly note several features of centralized key distribution that make it unsuitable for DPGs:

- A TTP that generates and distributes keys for a multitude of groups is a single point of failure and a likely performance bottleneck.

[1]Informally, this means that old keys cannot be known to new members and new keys cannot be known to former members.

- Since all group secrets are generated in one place, a TTP presents a very attractive attack target for adversaries. This is especially the case if a TTP serves as the key generation/distribution center for multiple groups.

- Environments with no hierarchy of trust are a poor match for centralized key transport. (For example, consider a peer group composed of members in different, and perhaps competing, organizations or countries.)

- Some DPG environments (such as *ad hoc* wireless networks) are highly dynamic and no group member can be assumed to be present all the time. However, most key distribution protocols assume fixed centers.

- It might be simply unacceptable for a single party to generate the group key. For example, each party may need assurance that the resulting group key is *fresh* and *random* (e.g., in case the key is later used for computing digital signatures).

- Achieving perfect forward secrecy (Def. 2.4.7 below) and resistance to known-key attacks (Def. 2.4.8 below) in an efficient manner is very difficult in the centralized key distribution setting.

Although we argue in favor of distributed, contributory key agreement for DPGs, we also recognize the need for a central point of control for group membership operations such as adding and deleting members. This type of a role (group membership controller) serves only to synchronize the membership operations and prevent chaos. However, the existence and assignment of this role is orthogonal to key establishment and is largely a matter of policy.

2.3 Previous Work

We now present several prior results in multi-party key agreement protocols. None of them provide any key authentication services and all are resistant only against passive attacks. Moreover, it is unclear how to add authentication services to these schemes.

A mathematically elegant scheme was proposed by Fiat et al. in [51]. A trusted center T selects a RSA-like modulus $n = pq$ and a *secret* element $\alpha \in \mathbb{Z}_n^*$ of large multiplicative order (such that it is hard to compute discrete logarithms). For any $1 \leq i \leq t$, each party M_i receives $\alpha^{x_i} \bmod n$ from T, with x_i random and relatively prime with x_j for $j \neq i$. In order to establish a secret group key S, each party M_i broadcasts x_i and then, after collecting all messages, computes $S = (\alpha^{x_i})^{x_1 \cdots x_{i-1} x_{i+1} \cdots x_t} \bmod n$ (i.e. a Diffie-Hellman key with all the contributions). Drawbacks of this protocol are that 1) it requires a trusted third party (T) and, 2) as shown in [51], two or more parties can collude and recover the secret α.

Another interesting scheme was presented in [100]. Given a $g \in \mathbb{Z}_p^*$ with p prime, each party M_i computes and broadcasts $y_i = g^{x_i} \bmod p$, where x_i is a randomly chosen secret. After receiving all the contributions, M_1 computes the group key:

$$S = g^{x_t g^{x_{t-1} g^{\cdots x_3 g^{x_2 x_1}}}}.$$

Note that to come up with the same group key, the other protocol parties needs to behave differently since the order of exponentiation is right to left, i.e., the protocol is asymmetric. In particular, for $j = 3, \ldots, t-1$, M_j sends $v_j = g^{v_{j-1}^{x_j}}$ to M_{j+1} (where $v_2 = y_1$) and then computes S. The party M_t simply waits v_{t-1} from M_{t-1} and then computes $S = v_{t-1}^{x_t}$.

One notable recent result is due to Burmester and Desmedt [29]. They construct a very efficient protocol (BD) which executes in only three rounds:

1. Each M_i generates its random exponent x_i and broadcasts $z_i = \alpha^{x_i}$.
2. Each M_i computes and broadcasts $W_i = (z_{i+1}/z_{i-1})^{x_i}$
3. Each M_i can now compute the key $S = z_{i-1}^{tx_i} \cdot W_i^{t-1} \cdot W_{i+1}^{t-2} \cdots W_{i-2} \bmod p$

The resulting key is $S = \alpha^{x_1x_2 + x_2x_3 + \cdots + x_tx_1}$. The protocol is proven secure provided the DH problem is intractable. However, there are some important assumptions underlying this protocol. Specifically, it requires each M_i to broadcast to the rest of the group and to receive $t-1$ messages in a single round. Moreover, the system has to handle t simultaneous broadcasts (in one round).

As mentioned in Section 2.1, Steiner et al. [103] introduced a class of protocols, called *generic* Group Diffie-Hellman (GDH) key agreement. This entire class has been proven resistant against passive attacks. In brief, [103] shows that *if a 2-party DH key is indistinguishable from a random value then a t-party DH key is also indistinguishable from a random value.* Later in this chapter, we describe in more detail some of the GDH protocols and extend them to provide authentication services.

2.4 Goals, Definitions and Notation

In addition to *key independence* alluded to above and resistance to all types of passive attacks, desired properties for a practical key agreement protocol typically include the following:

- Key Authentication
- Perfect Forward Secrecy (PFS)
- Resistance to Known-Key Attacks
- Key Confirmation and Key Integrity

All of these are necessary to achieve resistance to *active* attacks mounted by an increasingly powerful adversary. And, as always, ironclad security must be achievable with the lowest possible cost.

We now present some definitions for the above and other terminology used in this chapter. (Some of these are adapted from Menezes et al. [79])

Definition 2.4.1 *A* **key agreement protocol** *is a key establishment technique whereby a shared secret key is derived by two or more specified parties as a function of information contributed by, or associated with, each of these, such that no party can predetermine the resulting value.*

Definition 2.4.2 *A key agreement protocol is* **contributory** *if each party equally contributes to the key and guarantees its freshness.*

For example, according to this definition, the basic two-party Diffie-Hellman protocol is contributory. On the other hand, the ElGamal one-pass [79] protocol is not contributory as only one of the parties contributes a fresh exponent.

Definition 2.4.3 *Let $\mathcal{R}$ be an n-party key agreement protocol, $\mathcal{M}$ be the set of protocol parties and let S_n be a secret key jointly generated as a result of $\mathcal{R}$. We say that $\mathcal{R}$ provides* **implicit key authentication** *if each $M_i \in \mathcal{M}$ is assured that no party $M_q \notin \mathcal{M}$ can learn the key S_n (unless aided by a dishonest $M_j \in \mathcal{M}$).*

Definition 2.4.4 *A protocol provides* **key confirmation** *if a party is assured that its peer (or a group thereof) actually has possession of a particular secret key.*

Definition 2.4.5 *A contributory key agreement protocol provides* **key integrity** *if a party is assured that its particular secret key is a function of* **only** *the individual contributions of all protocol parties. In particular, extraneous contribution(s) to the group key cannot be tolerated even if it does not afford the attacker(s) with any additional knowledge.*

Definition 2.4.6 *An* **authenticated group key agreement protocol** *is a key agreement protocol which provides implicit key authentication.*

Definition 2.4.7 *A protocol offers* **perfect forward secrecy (PFS)** *if compromise of a long-term key(s) cannot result in the compromise of past session keys.*

Definition 2.4.8 *A protocol is said to be vulnerable to* **known-key attack** *if compromise of session keys allows: 1) a passive adversary to compromise keys of other sessions, or 2) an active adversary to impersonate one of the protocol parties. (See [28, 27], for details.)*

We will use the following notation:

n	number of protocol parties (group members)
i, j	indices of group members
M_i	i-th group member; $i \in [1, n]$
G	unique subgroup of $\mathbb{Z}_p^*$ of order q with p, q prime
q	order of the algebraic group
α	exponentiation base; generator[2] in group G
x_i	long-term secret key of M_i
r_i	random (secret) exponent $\in \mathbb{Z}_q$ generated by M_i
S_n	group key shared among n members
$S_n(M_i)$	M_i's view on a group key
K_{ij}	long-term secret shared by M_i and M_j, with $i \neq j$
$F()$	a function mapping elements from G to $\mathbb{Z}_q$

All arithmetic is performed in a cyclic group G of prime order q which is a subgroup of $\mathbb{Z}_p^*$ for a prime p such that $p = kq + 1$, for some integer k. In particular, we should select p as *safe* prime, i.e., $k = 2$.

No practical methods are known to compute partial information with respect to discrete logarithms (DL) in subgroup with this setting. Most DL-based schemes have been designed using a prime order subgroup. One of the advantages of working in such a group is that all the elements (except the unity element) are generators of the subgroup itself. Moreover, using subgroup of prime order seems to be a prudent habit [7]; it also results in increased efficiency.

When operating in subgroups it is important to take into account the attacks outlined in [7, 61]. To prevent masquerading or leaking of (even partial) information of the secret

[2] α can be computed by repeatedly selecting a random element $b \in \mathbb{Z}_p^*$ and computing $\alpha = b^{(p-1)/q} \bmod p$ until $\alpha \neq 1$.

values, each party has to verify that the purportedly random values it receives are in fact elements of the subgroup.[3]

Note that p, q and α are public and common to all users. Since they need to be generated only once (or very seldom), it is desirable to make the generation process unpredictable but verifiable to prevent the selection of weak or special primes. One approach is to use the NIST method for selecting DSA primes as described in the FIPS 186 document [84].

In this context, the ability of an active adversary C to modify or inject messages is quite "limited". In fact, any message m can be written as $m = \alpha^c \bmod p$, where α is a generator of the unique cyclic subgroup of $\mathbb{Z}_p^*$ having order q and c some exponent (perhaps unknown). Later on, we will suppose that the adversary C operates on this type of elements.

2.5 Authenticated 2-party Key Agreement

In this section we develop an extension to the Diffie-Hellman (DH) [49] key agreement method that provides key authentication. We explicitly avoid requiring any cryptographic tools (e.g., symmetric encryption or signatures) other than those necessary for plain DH key agreement.

Before turning to the actual protocol, it is important to emphasize that there exist secure protocols for authenticated DH-based key agreement. However, some are not contributory (such as ElGamal), some require more messages or assume a priori access to certified long-term keys, while others do not offer PFS or are vulnerable to so-called *known-key attacks.* (For example, some of the protocols in the MTI protocol family [78].) An additional goal is to come up with a protocol that is easily extendible from 2- to n-party key agreement. Yet another, perhaps superficial, issue has to do with minimizing the security dependencies of a protocol. For example, an authenticated DH-based key agreement can be easily constructed with the aid of conventional encryption. The security of the underlying protocol would then be dependent not only on the difficulty of, for example, the Diffie-Hellman Decision (DDH) problem (as far as key agreement) but also on the strength of the conventional encryption (as far as key authentication). Ideally, it should be possible to base all the security properties of a given protocol on a *single* hard problem such as the DDH problem in prime-order subgroups.

One protocol that satisfies the above criteria is A-DH, shown in Figure 2.1. It provides implicit key authentication as stated by the following theorem.

[3] Verifying the order of an element x by checking, for example, that $x^{(p-1)/q} \bmod p \neq 1$, is rather expensive. If p and q are carefully chosen such that the other prime factors of $\varphi(p)/2$ are close to the order of q, we can exclude elements of small order in an efficient manner by checking that $x^2 \neq 1 \bmod q$. Although this seems to be sufficient, the security of this method needs further study [61].

Protocol A-DH: Let p, q, G be as defined above, and let α be a generator of G.
Initialization. Let x_1 and x_2 be two integers such that $1 \leq x_1, x_2 \leq q-1$. Let M_1 and M_2 be two parties wishing to share a key and let $(x_1, \alpha^{x_1} \bmod p)$ and $(x_2, \alpha^{x_2} \bmod p)$ be the secret and public keys of M_1 and M_2, respectively. Thus, the public values of the system are $(p, q, \alpha, \alpha^{x_1}, \alpha^{x_2})$. The actual protocol is as follows:
Round 1:
M_1 selects random $r_1 \in_R \mathbb{Z}_q^*$,
$M_1 \longrightarrow M_2 : \alpha^{r_1} \bmod p$
Round 2:
M_2 selects random $r_2 \in_R \mathbb{Z}_q^*$, and computes
$K = F(\alpha^{x_1 x_2} \bmod p)$.
$M_2 \longrightarrow M_1 : \alpha^{r_2 K} \bmod p$
When M_1 receives $J = \alpha^{r_2 K} \bmod p$, computes $K^{-1} \bmod q$ and then $J^{r_1 K^{-1}} \bmod p$. The shared secret key is $S_2 = \alpha^{r_1 r_2} \bmod p$. Function $F()$ maps elements from G to $\mathbb{Z}_q$. If p is a safe prime $(p = 2q + 1)$ then a perfect mapping function would be $F(x)= x$ if $(x < q)$, $q - x$ if $(x >= q)$.

Figure 2.1: Authenticated Diffie-Hellman (A-DH)

Theorem 2.5.1 *The A-DH protocol is a contributory authenticated key agreement protocol.*

Proof: From the construction of the resultant session key $S_2 = \alpha^{r_1 r_2}$ it is evident that A-DH is *contributory*. In our model we assume that the two parties M_1 and M_2, wishing to share a secret, behave correctly. Moreover, we assume that any value computed as session key (S_2), by each party, is kept secret, i.e. it is infeasible for an attacker to obtain any (even partial) information about GK_2. Let C be an active adversary able to modify, delay, or inject messages. C's goal is to obtain the secret shared by M_1 and M_2, if any. There are four possible attack scenarios which we treat separately below. To prove our claim we use the following approach: if the attacker C were able to use a nondeterministic Turing machine TM to obtain information about the secret key, then the same TM could be used to solve an instance of the Diffie-Hellman Problem (DHP). The notation $TM(x_1, \ldots, x_n) = y$ means that, given input $x_1, \ldots, x_n$, TM outputs y.

1. **A-DH ends correctly (***passive attack***).** Suppose there exists a nondeterministic Turing machine TM_1 by which the attacker C is able to find the secret session key, i.e. $TM_1(\alpha^{r_1}, \alpha^{r_2 K}) = \alpha^{r_1 r_2}$. Since r_1 and r_2 are randomly chosen, it is easy to see that TM_1 could be used to solve a generic instance of DHP. In fact, given α^a and α^b, with unknown a, b, we would have $TM_1(\alpha^a, \alpha^{bK}) = \alpha^{ab}$.

2. C **substitutes α^{r_1} with α^x.** For simplicity's sake, we can assume that the attacker C knows the value x. The secret computed by M_1 is $\alpha^{r_1 r_2}$. The problem of computing this value has been treated in the previous case.

   The secret computed by M_2 is $\alpha^{x r_2}$. The easiest way to compute this value is extracting α^{r_2} from $\alpha^{r_2 K}$. But this is as hard as DH problem if K is the DH value of the public-keys of M_1 and M_2.

3. C **substitutes α^{r_2} with α^y.** Suppose the attacker C knows the value y. The secret computed by M_2 is $\alpha^{r_1 r_2}$. The hardness of computing this value has been already treated in previous cases. C can compute the M_1's secret, $\alpha^{y r_1 K^{-1}}$, trying to get the value $\alpha^{r_1 K^{-1}}$. However, computing $\alpha^{r_1 K^{-1}}$ is as hard as DHP if K is the DH value of the public-keys of M_1 and M_2.

4. C **substitutes α^{r_1}, α^{r_2} with α^x and α^y, respectively.** Since the A-DH protocol's messages are each other independent, this case is analogous to the cases 2 and 3.
   □

On top of implicit key authentication, a practical key agreement protocol must: 1) provide perfect forward secrecy and 2) be resistant to known-key attacks. These two properties are considered in the following theorems.

Theorem 2.5.2 *The A-DH protocol provides perfect forward secrecy (PFS).*

Proof: Suppose that the long-term key $K = F(\alpha^{x_1 x_2} \bmod p)$ is compromised. Then, an adversary knows both $\alpha^{r_1} \bmod p$ and $\alpha^{(r_2 K)K^{-1}} \equiv \alpha^{r_2} \bmod p$. Given these, computing the session key $S_2 = \alpha^{r_1 r_2} \bmod p$ is equivalent to solving the DH problem in prime-order subgroups. □

Theorem 2.5.3 *The A-DH protocol is resistant to known-key attacks.*

Proof: The scenario is the following: the attacker C is able to modify both the A-DH protocol's messages and then get the values computed by each party M_1 and M_2 as secret key. From these values, C tries to compute information by which he can impersonate one of the protocol parties. For the sake of simplicity, we can assume that the target of the attacker C is finding α^K by which he can impersonate M_2.

There are four cases to consider:

1. C knows α^{r_1}, $\alpha^{r_2 K}$ and $\alpha^{r_1 r_2}$ (*passive known-key attack*).

2. C substitutes α^{r_1} with α^x, then knows α^{r_1}, $\alpha^{r_2 K}$, $\alpha^{r_1 r_2}$ and α^{r_2}.

3. C substitutes α^{r_2} with α^y, then knows α^{r_1}, $\alpha^{r_2 K}$, $\alpha^{r_1 r_2}$ and $\alpha^{r_1 K^{-1}}$.

4. C substitutes both α^{r_1}, α^{r_2} with α^x and α^y, respectively. Then C knows α^{r_1}, $\alpha^{r_2 K}$, α^{r_2} and $\alpha^{r_1 K^{-1}}$.

Therefore, the general problem that C has to solve is: given α^a and α^{ab} getting α^b where a, b are unknown random values[4]. It easy to see that, when working in subgroup of prime order, this problem is equivalent to DH problem. □

A nice feature of the A-DH protocol is that it does not require *a priori* knowledge of the long-term public keys of the parties involved. In fact, certificates can be piggy-backed onto existing protocol messages. This is a consequence of the protocol's "asymmetry".

2.6 Authenticated Group Key Agreement

In [102], a class of *generic n-party DH protocols* is defined. The entire protocol class is shown secure against passive adversaries based on the intractability of the Diffie-Hellman Decision (DDH) problem. Several concrete protocols were demonstrated that fit the requirements of DPGs. Moreover, these protocols are shown to be optimal with respect to certain measures of protocol complexity [102, 18]. In this section we extend the GDH protocols to provide implicit key authentication. In doing so, we make use of the A-DH protocol discussed in section 2.5.

2.6.1 Authenticated GDH.2 protocol

Two practical protocols: GDH.2 and GDH.3 are defined in [102]. (Another protocol, GDH.1, is used for demonstration purposes only.) The GDH.2 protocol is minimal in terms of the total number of protocol messages. GDH.3, on the other hand, aims to minimize computation costs. Although, the discussion below focuses on extending GDH.2, we note that all of the techniques we consider are easily adapted to GDH.3.

We begin with a brief overview of GDH.2 in Figure 2.2. This basic protocol can be easily amended to provide implicit key authentication in an efficient manner. This variation (A-GDH.2, shown in Figure 2.3) differs from the basic protocol only in the last round, hence we are only concerned therewith.

We assume that M_n shares (or is able to share) with each M_i a distinct secret K_{in}.

[4] For example, if this problem were easy, the attacker C could get α^K from the values α^{r_2}, $\alpha^{r_2 K}$.

Protocol GDH.2: Let $\mathcal{M} = \{M_1, \ldots, M_n\}$ be a set of users wishing to share a key S_n. The GDH.2 protocol executes in n rounds. In the first stage ($n-1$ rounds) contributions are collected from individual group members and then, in the second stage (n-th round) the group keying material is broadcast. The actual protocol is as follows:

Initialization. Let p be a prime and q a prime divisor of $p-1$. Let G be the unique cyclic subgroup of $\mathbb{Z}_p^*$ of order q, and let α be a generator of G.

Round i $(0 < i < n)$**:**

1) M_i selects random $r_i \in_R \mathbb{Z}_q^*$.

2) $M_i \longrightarrow M_{i+1}$: $\{\alpha^{\frac{r_1 \cdots r_i}{r_j}} | j \in [1, i]\}, \ \alpha^{r_1 \cdots r_i}$

Round n**:**

1) M_n selects random $r_n \in_R \mathbb{Z}_q^*$.

2) $M_n \longrightarrow$ ALL M_i: $\{\alpha^{\frac{r_1 \cdots r_n}{r_i}} | i \in [1, n[\}$

Figure 2.2: Group Diffie-Hellman (GDH.2)

For example, we can set $K_{in} = F(\alpha^{x_i \cdot x_n} \bmod p)$ with $i \in [1, n-1]$. Where x_i is a secret long term exponent selected by every M_i $(1 \leq x_i \leq q-1)$ and $\alpha^{x_i} \bmod p$ is the corresponding long-term public key of M_i.

Protocol A-GDH.2:

Rounds 1 **to** $n-1$**:** identical to GDH.2

Round n**:**

1) M_n selects random $r_n \in_R \mathbb{Z}_q^*$

2) $M_n \longrightarrow$ ALL M_i: $\{\alpha^{\frac{r_1 \cdots r_n}{r_i} \cdot K_{in}} | i \in [1, n[\}$.

Upon receipt of the above, every M_i computes: $\alpha^{(\frac{r_1 \cdots r_n}{r_i} \cdot K_{in}) \cdot K_{in}^{-1} \cdot r_i} = \alpha^{r_1 \cdots r_n} = S_n$.

Figure 2.3: Authenticated Group Diffie-Hellman (A-GDH.2)

In this protocol, each group member obtains an (implicitly) authenticated shared key with M_n. Moreover, if we trust M_n to behave correctly, a group member can also be sure the key shared with M_n is the same key M_n shares with all other members.

Theorem 2.6.1 *A-GDH.2 is a contributory authenticated key agreement protocol.*

Proof (sketch): From the construction of the resultant session key $S_n = \alpha^{r_1 \cdots r_n}$ it is evident that A-GDH.2 is *contributory.*

Let C be an active adversary who can modify, delay, or inject messages. C's goal is to share a key with either M_i, for $i \in [1, n[$, or with M_n by masquerading as some M_i. In case of the former, all considerations of the proof in theorem 2.5.1 apply.

Assume that C wants to masquerade as M_i. Let $S_n(M_n)$ be the key computed by M_n. It can be expressed as:

$$S_n(M_n) = \alpha^{c_n \cdot r_n}$$

where c_n is a quantity possibly known to C, i.e., in round $n-1$ C can replace $\alpha^{r_1 \cdots r_{n-1}}$ with α^{c_n} in the message from M_{n-1} to M_n. C can also replace the other $(n-1)$ values in the same message:

$$\alpha^{\frac{r_1 \cdots r_{n-1}}{r_j}} \; (j \in [1, n[) \rightarrow \alpha^{c_j}$$

for some known c_j-s. This will cause M_n to output in the last round:

$$\{ \alpha^{c_j \cdot r_n \cdot K_{jn}} \mid j \in [1, n[\}$$

Now, since C knows all c_j, she also knows (or can easily compute) all c_j^{-1}. Hence, C can compute:

$$\{ \alpha^{r_n \cdot K_{jn}} \mid j \in [1, n[\}$$

However, extracting information of $S_n(M_n)$ is intractable if the DDH problem in prime-order subgroup is hard. □

Theorem 2.6.2 *The A-GDH.2 protocol provides perfect forward secrecy.*

Proof: Suppose that all long-term keys $\{K_{in} \mid i \in [1, n[\}$ are compromised. Then, our adversary is able to compute a subset of $V = \{\alpha^{\Pi(\mathcal{S})} | \mathcal{S} \subset \{r_1, \ldots, r_n\}\}$. But, as shown in [102], given V, it is intractable to find information on the group key $S_n = \alpha^{r_1, \ldots r_n}$, if the DDH problem in prime-order subgroup is hard. □

2.6.2 Resistance to known-key attacks.

A-GDH.2 is resistant to *passive* known-key attacks since the session keys do not contain any long-term information. Resistance to *active* known-key attacks, on the other hand, is somewhat dubious for reasons stated below.

Let $S_n(M_i)$ be the session key computed by each M_i. For $0 < i < n-1$ we can re-write it as $\alpha^{c_i r_i K_{in}^{-1}}$. For M_n, $S_n(M_n) = \alpha^{c_n r_n}$ where c_i is a quantity possibly known to the adversary C. C also knows a subset of $\{\alpha^{\Pi(\mathcal{S})} | \mathcal{S} \subset \{r_1, \ldots, r_n\}\}$. Using these to find $\alpha^{K_{in}}$ or $\alpha^{K_{in}^{-1}}$ (for $1 \leq i \leq n-1$), is intractable if the DDH problem in prime-order subgroup is hard.

Despite the above, some forms of active known-key attacks are possible. Suppose, for example, that C tries to impersonate M_1. It starts by sending α^{c_1} to M_1 in the last protocol round (where c_1 is selected by C). As a result, M_1 computes: $S_n(M_1) = \alpha^{c_1 r_1 K_{1n}^{-1}}$. Since this key is corrupted (i.e., not shared with any other M_i), we can assume that M_1 will detect the problem and re-run the protocol. Suppose further that C somehow manages to discover this malformed key.[5] In the next protocol run, C can substitute the message from M_{n-1} to M_n with:

$$\alpha^{c_1 r_1 K_{1n}^{-1}}, \ldots, \alpha^{c_1 r_1}$$

In other words, C substitutes only the first and the last sub-keys in the flow; the rest of the values are unchanged. This causes M_n to compute $S_n(M_n) = (\alpha^{c_1 r_1})^{r_n}$. M_n will also compute (as a sub-key for M_1):

$$(\alpha^{c_1 r_1 K_{1n}^{-1}})^{r_n K_{1n}} = \alpha^{c_1 r_1 r_n}$$

and will broadcast this value in the last protocol round. The end-result is that C shares a key with M_n.

There are a few issues with this type of attack. First, it relies on the lack of key confirmation which we discuss later in the chapter. Second, it does not fit the usual definition of a known-key attack since C is only able to share a key with M_n, not with the rest of the group. (We note that known-key attacks were only defined in the context of 2-party protocols. Their definition in a group setting remains to be worked out.) Also, as noted in [27], a simple cure for known-key attacks is by setting $S_n = h(S_n(M_i))$ where $h()$ is an appropriate collision-resistant hash function such as SHA [85].

2.6.3 Complete (Strong) Group Key Authentication

The above protocol (A-GDH.2) achieves implicit key authentication in a relatively *weak* form since the key is not directly authenticated between an arbitrary M_i and M_j ($i \neq j$). Instead, all key authentication is performed through M_n. This may suffice in some environments, e.g., when the exact membership of the group is not divulged to the individual M_i's. Another reason may be that M_n is an entity trusted by all other members, e.g., M_n is an authentication server.

According to Definition 3.3, A-GDH.2 will result in all participants agreeing on the same key if we assume M_n behaves correctly. However, no one – including M_n – can be sure of other members' participation. In fact, one or more of the intended group members may be "skipped" without detection. Also, a dishonest M_n could partition the group into two without detection by group members. On the one hand, we assume a certain degree of trust

[5] This assumption is what makes active known-key attacks very unlikely in practice.

in all group members (including M_n), e.g., not to reveal the group key to outsiders. On the other hand, one might want to limit this trust when it comes to group membership, i.e., M_n might not be universally trusted to faithfully include all (and only) group members.

In more concrete terms, our failure model is based on:

> A malicious insider (group member) seeking to alter the group membership by excluding some members – possibly including itself – from participation in key agreement. For example, this may translate into attempting to physically circumvent certain group members or corrupting intermediate values that subsequently contribute to the excluded members' keys.

On the other hand, our failure model specifically **excludes**:

> A malicious insider revealing the group key or any other group (or its own) secrets to outsiders.
>
> An insider (malicious or otherwise) exhibiting any other form of anomalous behavior.

In order to clarify the above, we introduce the following feature:

Definition 2.6.3 *Let $\mathcal{R}$ be an n-party key agreement protocol and $\mathcal{M}$ be a set of protocol parties (DPG). We say that $\mathcal{R}$ is a* **complete group key authentication** *protocol if, for every i, j $(0 < i \neq j \leq n)$ M_i and M_j compute the same key $S_{i,j}$* **only if** *$S_{i,j}$ has been contributed to by every $M_p \in \mathcal{M}$. (Assuming that M_i and M_j have the* **same view** *of the group membership.)*

An alternative definition for complete group key authentication is as authenticated group key agreement for all (M_i, M_j) pairs $(0 < i \neq j \leq n)$.

A-GDH.2 can be augmented to provide complete group key authentication as shown in Figure 2.4. (To better illustrate SA-GDH.2 and its differences with respect to A-GDH.2, a 4-party example is shown in Figure 2.5.)

The biggest change in the present protocol, SA-GDH.2, is the requirement for *a priori* availability of all members' long-term credentials. In effect, each M_i is required to have two shared keys (one in each direction) with every other M_j. For every distinct *ordered* pair $< i, j >$ $(0 < i \neq j \leq n)$ let $< K_{ij}, K_{ij}^{-1} >$ denote the *unidirectional* key shared by M_i and M_j and its inverse, respectively. Although it may appear otherwise, individual key inverses of the form K_{ij}^{-1} **do not** need to be computed (see below).

Drawbacks: SA-GDH.2 is clearly more expensive than A-GDH.2. First, it requires $n-1$ exponentiations from every M_i during stage 1 as opposed to i in A-GDH.2. Second, if pairwise keys (K_{ij}) are not pre-computed, as many as $(n-1)$ additional exponentiations must be performed. Note that in the last round, only one exponentiation is done since M_i can pre-compute the value: $(K_{i1}^{-1} \cdots K_{in}^{-1}) \cdot r_i$ immediately following the i-th round.

Advantages: unlike A-GDH.2, SA-GDH.2 allows each member to be explicitly aware of the exact group membership. This may be desired in some environments. Also, the protocol is computationally symmetric, i.e., each member performs the same sequence of computational steps and the same number of exponentiations.

Theorem 2.6.5 *SA-GDH.2 offers complete group key authentication.*

Proof (sketch): Suppose M_i and M_j compute the same key while following the protocol correctly. Let $K_n = S_n(M_i) = S_n(M_j)$ and, suppose also, that some $M_p \in \mathcal{M}$, $(p \neq i, j)$ has not contributed to this key. Let V_i, V_j denote the values received by M_i and M_j, respectively, in the last round of the protocol. Recall that:

$$S_n(M_i) = (V_i)^{(K_{1i}^{-1} \cdots K_{ni}^{-1}) \cdot r_i}$$

and, similarly:

$$S_n(M_j) = (V_j)^{(K_{1j}^{-1} \cdots K_{nj}^{-1}) \cdot r_j} =$$

Since all other group members have contributed to the key, we can re-write V_i as (V_j is similar):

$$V_i = \alpha^{(\frac{r_1 \cdots r_n}{r_p r_i}) \cdot (\frac{K_{1i}^{-1} \cdots K_{ni}^{-1}}{K_{pi}^{-1}})}$$

Then,

$$S_n(M_i) = \alpha^{(\frac{r_1 \cdots r_n}{r_p}) \cdot K_{pi}^{-1}}$$

which must equal:

$$S_n(M_j) = \alpha^{(\frac{r_1 \cdots r_n}{r_p}) \cdot K_{pj}^{-1}}$$

However, this is impossible since K_{pi}^{-1} and K_{pj}^{-1} are distinct and secret values. $\square$

Remark 2.6.6 *An interesting feature of SA-GDH.2 is its resistance to known-key attacks. Although we do not treat this topic in detail, it can be easily observed that an attack of the sort described in Section 2.6.1 cannot succeed against SA-GDH.2.*

2.7 Efficiency Summary

We now consider the costs incurred by the protocols described above. The following two tables summarize, respectively, the communication and computation overhead of the following:

- GDH.2 – plain group key agreement [102].
- A-GDH.2 - authenticated group key agreement as specified in Section 2.6.1. Long-term keys K_{in} are assumed to be pre-computed.
- A-GDH.2* - same as A-GDH.2 but long-term keys K_{in} are computed as part of the protocol; this also implies that public exponents of all group members must be accumulated in the course of the protocol.
- SA-GDH.2 – complete group key authentication

The first table illustrates the communication, and the second computation, costs. The latter is broken down into exponentiation, inverse computation and multiplication. Exponentiation is clearly the costliest operation as it requires $O(log^3 p)$ bit operations in $\mathbb{Z}_p^*$. Given a and p, finding the inverse of $a \in \mathbb{Z}_p^*$ requires only $O(log^2 p)$ bit operations (using the extended Euclidean algorithm). Similarly, the multiplication of a and b modulo p requires $O(log^2 p)$ bit operations. (See [68], [79] for a complete treatment of modular operations.)

	Protocols:			
Communication Costs:	GDH.2	A-GDH.2	A.GDH.2*	SA-GDH.2
rounds	n	n	n	n
broadcasts	1	1	1	1
total msgs	n	n	n	n
total bandwidth	$(n^2+n)/2-1$	$(n^2+n)/2-1$	n^2	n^2
msgs sent per M_i	1	1	1	1
msgs received per M_i	2	2	2	2

The only somewhat surprising element of this analysis is the relatively low additional cost of SA-GDH.2 as compared to that of GDH.2 and A-GDH.2. Considering that it offers complete group key authentication and several other useful services (when coupled with key confirmation; see below) the added overhead is well justified.

Comput. Costs:	Protocols: GDH.2	A-GDH.2	A.GDH.2*	SA-GDH.2
exponen. for M_i	$i+1$	$i+1$	$i+2$	n
exponen. for M_n	n	n	$2n-1$	n
total exponen.	$(n^2+3n)/2-1$	$(n^2+3n)/2-1$	$(n^2+4n)/2-2$	n^2
inverses for M_i			1	1
inverses for M_n				1
total inverses			$n-1$	n
multipl. for M_i		1	1	$2n-2$
multipl. for M_n		$n-1$	$n-1$	$2n-2$
total multipl.		$2n-2$	$2n-2$	$2n^2-2n$

2.8 Dynamic Group Changes

Thus far, we treated groups as static entities. However, more often than not, group changes during the lifetime of a group. New members join in and old members leave (or get evicted from) the group. Moreover, due to "environmental" factors such as network failures groups can be partitioned. Similarly, when network partitions heal, multiple groups need to merge into one.

Therefore, the problem is how to share a secret key (achieve key agreement) in the face of membership changes. As stated in the introduction we require *key independence* and therefore need to compute a new (authenticated and contributory) key. Intuitively, we can approach the problem in two ways:

- **Starting from scratch:** all group members start the key agreement protocol anew and destroy any and all old values.
- **Using previous information:** in order to save computations, recycle old information to the extent possible (and secure) to share a common secret.

It seems clear that the first approach is expensive, unscalable and utterly unsuitable for environments with frequent membership changes.

In [103] it is showed how the underlying non-authenticated Group Diffie-Hellman key agreement protocols can be extended to achieve very efficient and flexible dynamic operation, i.e. adding/deleting a single member, fusion/fission of sub-groups and re-keying.

[6]Note that any group member can take that role; the decision who is group controller is simply a matter of policy.

As an example, Figure 2.6 illustrates the (unauthenticated) member addition protocol. The trick in that protocol (as well as in all the other AKA protocols) is the caching by the group members of the keying information (i.e., partial keys) distributed in the broadcast phase of the most recent IKA/AKA protocol run. This information is incrementally updated for an AKA protocol run and re-distributed to the new incarnation of the group.

Figure 2.7 describes the authenticated version of the member addition protocol. The first flow is exactly the same as in the unauthenticated case except that we use now the broadcast flow of the authenticated protocols which also contain long-term keys K_{in} in the exponent. The Round 2 is essentially the same as the second round of the normal A-GDH.2 protocol and guarantees implicit key authentication. As defined in Figure 2.7 each member would have to remember the sequence of AKA protocols and the corresponding group controllers to be able to factor out the long-term keys K_{jk}. However, if M_i substitutes his key contribution r_i with $r_i \cdot K_{in}^{-1}$ after each broadcast flow, there is no need to save this history-information and computing the key is exactly the same operation as for IKA protocols.

The extension from unauthenticated to authenticated AKA protocols can be performed in the same straight-forward manner also for all other AKA protocol described in [103] and will be omitted in the sequel.

2.9 Group Key Agreement and Byzantine Agreement

Group key agreement (GKA), in general, has similarities to the well-known Byzantine agreement (BA) problem ([73]) but there are a number of distinguishing features. The fault model in GKA is not Byzantine since we certain degree of trust is assumed among the group members, e.g., not to reveal the group key.

The standard BA requirements are: agreement, validity and termination. The validity requirement usually means: if all honest participants have the same input then they will agree on that value, otherwise they will agree on an an arbitrary value. Although termination and agreement would be required by complete authenticated key agreement too, the validity requirement is quite different, namely that the agreement is private to the participants[7] and that it is both fresh and random. Therefore, we claim that BA alone is not enough to build a robust GKA protocol.

On the other hand, GKA has similarities with secure multiparty computation (SMPC, e.g [38, 41]). In fact, GKA can be viewed as a special case of SMPC. However, we note that general SMPC techniques typically yield highly inefficient protocols.

[7]Note that BA protocols in general do not care about confidentiality.

Despite the above, BA could be used for key confirmation but that would represent overkill: BA protocols in the best-possible settings (signatures) require at least $(t+1)$ rounds to tolerate t failures. If we set $t = 0$ (since we do not worry about Byzantine faults) we still need a parallel broadcast of n signatures which is rather costly.

2.10 On Tree-based Approaches

Quite recently key agreement protocols, with very low communication cost, have been proposed. Generally, it is assumed that group members are nodes of a tree structure. The resulting group key is a series of exponentiations with a fixed base, α say, repeated many times as function of the number of group members. For example, a typical key structure looks like $\alpha^{\alpha^{\cdots^{\alpha^k}}}$, where k is the actual secret and the number of exponentiations is a (logarithmic) function of the number of group members.

Unfortunately, this key structure is flawed and thus all these schemes might easily be broken. Recovering the secret k becomes easier as the number of group members grows. This is because the order of an element $g \in G$ is smaller than the order of G itself and this is also valid for each exponent. For example, it may happen that k is taken modulo 2, hence the attacker could simply guess which between 0 and 1 is k. There is no ideal mapping function that could work and acceptable solutions may require to multiply each exponent with some random secret as done in the scheme [100], shortly described in Section 2.3.

2.11 Summary

In conclusion, this chapter presented new definitions and protocols geared for the dynamic peer group (DPG) settings. In particular, it showed how key authentication can be incorporated into a particular class of group key agreement protocols. Later, chapter 4 shows how to add other important security services such as key confirmation and entity authentication.

Protocol SA-GDH.2:

Round i $(0 < i < n)$**:**

1) M_i receives a set of n intermediate values: $\{V_k | 1 \leq k \leq n\}$. ($M_1$ which can be thought of as receiving an empty set in the first round):

$$V_k = \begin{cases} \alpha^{(\frac{r_1 \cdots r_{i-1}}{r_k}) \cdot (K_{k1} \cdots K_{k(i-1)})} & \textbf{if } k \leq (i-1) \\ \alpha^{(r_1 \cdots r_{i-1}) \cdot (K_{k1} \cdots K_{k(i-1)})} & \textbf{if } k > (i-1) \end{cases}$$

2) M_i updates each V_k as follows:

$$V_k = \begin{cases} (V_k)^{K_{ik} \cdot r_i} = \alpha^{(\frac{r_1 \cdots r_i}{r_k}) \cdot (K_{k1} \cdots K_{ki})} & \textbf{if } k < i \\ (V_k)^{K_{ik} \cdot r_i} = \alpha^{(r_1 \cdots r_i) \cdot (K_{k1} \cdots K_{ki})} & \textbf{if } k > i \\ V_k & \textbf{if } k = i \end{cases}$$

Remark 2.6.4 In the initial round M_1 sets $V_1 = \alpha^1$.

Round n**:**

1) M_n broadcasts a set of all V_k values to the group.

2) On receipt, each M_i selects the appropriate V_i where:

$$V_i = \alpha^{(\frac{r_1 \cdots r_n}{r_i}) \cdot (K_{1i} \cdots K_{ni})}$$

M_i proceeds to compute:

$$(V_i)^{(K_{1i}^{-1} \cdots K_{ni}^{-1}) \cdot r_i} = \alpha^{r_1 \cdots r_n}$$

For the above, instead of computing $n-1$ individual key inverses of the form K_{ji}^{-1}, each M_i computes only a single *compound* inverse $P_i^{-1} = (K_{1i}^{-1} \cdots K_{ni}^{-1})$ where $P_i = (K_{1i} \cdots K_{ni})$

Figure 2.4: Group Diffie-Hellman with Complete Key Authentication (SA-GDH.2)

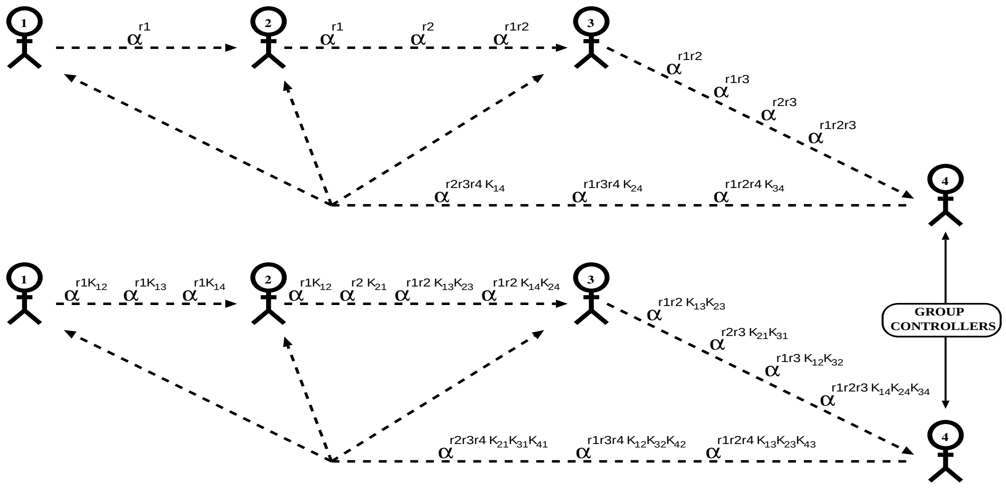


Figure 2.5: An example/comparison of A-GDH and SA-GDH.2

Protocol GDH.2-MA: Let M_{n+1} be the member to be added and M_n the group controller[6].

Round 1**:**

1) M_n selects random $\widehat{r_n} \in_R \mathbb{Z}_q^*$

2) $M_n \longrightarrow M_{n+1}$: $\{\alpha^{\widehat{r_n}\frac{r_1\cdots r_n}{r_i}} | i \in [1,n]\},\ \alpha^{\widehat{r_n} r_1 \cdots r_n}$

Round 2**:**

1) M_{n+1} selects random $r_{n+1} \in_R \mathbb{Z}_q^*$.

2) $M_{n+1} \longrightarrow M_i$: $\{\alpha^{\widehat{r_n}\frac{r_1\cdots r_{n+1}}{r_i}} | i \in [1,n+1]\}$

Upon receipt of above, every M_i computes the new key as usual and stores last flow for future AKA protocol runs. Additionally, M_n replaces his key contribution r_n with $\widehat{r_n} r_n$.

Figure 2.6: Member Addition Protocol for GDH.2

Protocol A-GDH.2-MA:

Round 1**:**

1) M_n selects random $\widehat{r_n} \in_R \mathbb{Z}_q^*$.

2) $M_n \longrightarrow M_{n+1}$: $\{\alpha^{\widehat{r_n}\frac{r_1 \cdots r_n}{r_i} \cdot K_{in}} | i \in [1, n]\}, \ \alpha^{\widehat{r_n} r_1 \cdots r_n}$

Round 2**:**

1) M_{n+1} selects random $r_{n+1} \in_R \mathbb{Z}_q^*$.

2) $M_{n+1} \longrightarrow M_i$: $\{\alpha^{\widehat{r_n}\frac{r_1 \cdots r_{n+1}}{r_i} \cdot K_{in} K_{in+1}} | i \in [1, n+1]\}$,

Upon receipt of above, every M_i computes the new key as $\alpha^{(\widehat{r_n}\frac{r_1 \cdots r_n}{r_i} \cdot K_{in} K_{in+1}) \cdot K_{in}^{-1} K_{in+1}^{-1} \cdot r_i} = \alpha^{\widehat{r_n} r_1 \cdots r_{n+1}} = S_{n+1}$. and stores last flow for future AKA protocol runs. Additionally, M_n replaces his key contribution r_n with $\widehat{r_n} r_n$.

Figure 2.7: Member Addition Protocol for A-GDH.2

Chapter 3

Group Signature Schemes

Group signatures allow any member of a potentially large group to sign on behalf of the group. Group signatures are anonymous and unlinkable for everyone with the exception of a designated group manager who can co-relate signatures and reveal the identity of the actual signer. At the same time, no one (including a group manager) can misattribute a valid group signature. Group signatures are claimed to have many practical applications in e-commerce as well as in military and legal fields. Despite some interesting and eclectic results, group signatures remain confined to academic literature. The focus of this chapter is three-fold. First, it discusses certain issues that stand in the way of practical applications of group signatures and uses the example of recent group signature schemes to illustrate certain problems. Second, it presents a candidate scheme whose security is based on a widely accepted number-theoretic assumption. The several results described in this chapter have appeared in [9, 12, 15, 16].

3.1 Introduction

Group digital signatures are a relatively new concept introduced by Chaum and van Heijst [43] in 1991. A group signature, akin to its traditional counterpart, allows the signer to demonstrate knowledge of a secret with respect to a specific document. A group signature is publicly verifiable: it can be validated by anyone in possession of a group public-key. However, group signatures are anonymous in that no one, with the exception of a designated group manager, can determine the identity of the signer. Furthermore, group signatures are unlinkable which makes it computationally hard to establish whether or not multiple signatures are produced by the same group member. In exceptional cases (such as a legal dispute) any group signature can be "opened" by a group manager to reveal unambiguously the identity of the actual signer. At the same time, no one —iincluding 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 [44]. 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. The group property offers the further advantage of concealing the identity of the cash-issuing bank [74].

The rest of this chapter is organized as follows. In the next section, we formally define group signatures and the associated security properties. Section 3.6 discusses the difficulty of coalition-resistance. As an educational exercise, we review in Section 3.7 our (failed) attempts to design a coalition-resistant group signature scheme. We then present a new candidate scheme with security based on the strong RSA assumption.

3.2 Definitions

We now present a more formal definition and desired properties of group signatures. For an in-depth discussion of this subject, we refer the reader to [32].

Definition 3.2.1 *A* group signature scheme *is a digital signature scheme comprised of the following procedures:*

- `SETUP`*: An algorithm for generating the initial group public-key $\mathcal{Y}$.*
- `JOIN`*: A protocol between the group manager and a user that results in the user becoming a new group member.*
- `SIGN`*: A protocol between a group member and a user whereby a group signature on a user-supplied message is computed by the group member.*
- `VERIFY`*: An algorithm for establishing the validity of a group signature given a group public-key and a signed message.*
- `OPEN`*: An algorithm that, given a signed message and a group secret key, determines the identity of the signer.*

A secure group signature scheme must satisfy the following properties:

- **Unforgeability**: Only group members are able to sign messages on behalf of the group.
- **Anonymity**: Given a signature, identifying the actual signer is computationally hard for everyone but the group manager.
- **Unlinkability**: Deciding whether two different signatures were computed by the same group member is computationally hard.
- **Exculpability**: Neither a group member nor the group manager can sign on behalf of other group members.[1]

  A related definition is that of *framing* [44] which captures the property that a group member is made responsible for a signature he did not produce.
- **Traceability**: The group manager is able to open a signature and identify the actual signer; moreover, a signer cannot prevent the opening of a valid signature.

The emphasis of the previous properties is on the fact that a *single* group member cannot generate signatures that are (in case of a dispute) not traceable to him by the group manager. This means that a subset of group members, pooling together their secrets, can generate valid group signatures that cannot be opened by the group manager. In order to avoid this we need the following property:

- **Coalition-resistance**: A colluding subset of group members cannot generate a valid signature that cannot be traced. This property will be discussed in more detail in Section 3.6.

Definition 3.2.2 *The* efficiency *or* practicality *of a group signature scheme is typically based on the following parameters:*

- *the size of the group public-key $\mathcal{Y}$;*
- *the size of a group signature;*
- *the efficiency of* `SIGN`*,* `VERIFY` *and* `OPEN`*; and*
- *the efficiency of* `SETUP` *and* `JOIN`*.*

[1] Note that the above does not preclude the group manager from creating fraudulent signers (i.e., nonexistent group members) and then producing group signatures.

3.3 Previous Work

As mentioned in the introduction, *group signatures*[2] were first introduced and realized by Chaum and van Heijst [43] in 1991. A number of improvements and enhancements followed. Most previous results exhibit some or all of the following shortcomings:

1. The size of the group public-key depends on the size of the group.
2. The length of a group signature depends on the size of the group.
3. New member addition requires restarting the entire system or involves re-issuing all members' keys and changing the group public-key.
4. Exclusion/revocation of group members requires re-issuing all members' keys and changing the group public-key.

In [44], Chen and Pedersen answer some open questions raised in [43] and realize the first constructions which allow new members to dynamically join the group. Camenisch [31] introduces and constructs *generalized group signatures* which allow some coalitions of group members to sign on the group's behalf. However, all these signature schemes present the main drawback that the length of the group public-key depends on the number of group members. Note that this is always the case when *unconditional* anonymity is desired [45].

Based on previous work of Park, I. Lee, and Won [88] (flawed in [77]), Kim, Park, and Won [67] put forth the concept of *convertible group signatures.* In addition to the properties of the standard group signatures, convertible group signatures of a given group member can be turned into ordinary signatures if that member releases some secret information. Later, Lim and P. J. Lee [72] (see also [77]) broke their schemes, showing that the parameters were improperly chosen and, more importantly, that the proposed group signatures were not coalition-resistant. Wang, Hwang, and N.-Y. Lee [107] further comment that their schemes do not allow to uniquely identify the original signer, even if the signer honestly follows the signature procedure. The reverse direction, i.e., converting an ordinary signature into a group signature, was investigated by Petersen [90]. His generic conversion makes use of the 'indirect disclosure proof' technique [53]; the length of the resulting group signatures is, however, linear in the number of group members.

Following Shamir [97], Park, Kim, and Won [87] (see also [89]) suggest *ID-based group signatures.* Their scheme was broken by Mao and Lim [75]: exploiting the prime order subgroup structure of the scheme, they showed that the anonymity of the signatures was

[2] The notion of group signatures is sometimes confused with the notion of *group-oriented signatures* where certain subsets of a group of people are allowed to sign on behalf of the group [48]. However, these latter schemes do not provide a method for identifying the signers.

not guaranteed. Moreover, the Park-Kim-Won scheme suffers from being rather expensive and 'static' in the sense that if new group members are added, the previously signed messages can no longer be verified with the updated public-key. A much better ID-based group signature scheme which does not present these limitations was proposed by Tseng and Jan [104]. In [106], Tseng and Jan also propose a group signature scheme based on the related notion of self-certified public keys [57]. See Section 3.6 for a discussion on the security of these two schemes.

Another efficient group signature scheme was recently proposed by W.-B. Lee and Chang [71], but, unfortunately, it does not enjoy the desirable property of unlinkability. An improved (i.e., unlinkable) version of the Lee-Chang scheme is proposed in [105]. Actually, any obvious attempt to make the Lee-Chang scheme unlinkable would likely fail; furthermore, both the original and the improved Lee-Chang schemes are known to be susceptible to *universal* forgeries [63].

In [31] an efficient group signature scheme was presented, providing computational anonymity, ability to add (or remove) group members after the initial setup, and the possibility of being generalized by allowing authorized set of group members to sign collectively (appearing to verifiers as a single signer) on behalf of the group. This scheme can be extended to allow the functionality of the group manager to be shared among several entities. The drawbacks include the size of the public-key and the signature size (both proportional to group size).

The first group signature suitable for *large groups* is that of Camenisch and Stadler [37]: both the length of the group public key and the group signatures are independent of the group's size. Moreover, a new member addition does not involve re-issuing other members' keys and/or changing the group public-key. The Camenisch-Stadler scheme was subsequently improved by Camenisch and Michels in [33] (see also [34]).

Lysyanskaya and Ramzan introduce *group blind signatures* in [74]. These signatures require that a group member signs on group's behalf a document without knowing its content. In [16], two other extensions of group signatures are discussed, namely *multi-group signatures* and *sub-group signatures*. A multi-group signature is a set of regular group signatures generated by the same signer as a member of multiple groups, whereas a sub-group signature is a group signature generated by a sub-group of group members. The main difference with the generalized group signatures as defined by Camenisch [31] resides in that the size of the sub-group is publicly verifiable.

3.4 What Stands in the Way?

Schemes such as [37, 33] make group signatures more attractive and bring the entire concept closer to practice. While there is still no single *killer app*, a number of applications can be envisaged for group signatures. Examples include:

- Government, military and commercial press releases
- Email messages in anonymous support groups
- Electronic cash (with banks forming a cash-issuing group)
- Issuance of stock, bond and other financial reports
- Voting applications
- Document signing by certified (but anonymous) professionals, e.g., lawyers, notaries public, auditors, etc.

There remain, however, some bothersome issues that stand in the way of real-world applications and deployments of group signatures. In this chapter we concentrate on two that we believe to be the most important:

- Security against coalitions
- Member deletion (revocation) complexity

The first problem is straight-forward. If a scheme is vulnerable to coalition attacks, its utility is limited to environments where member collusion is not a threat. Although such environments exist[3], in general, coalition resistance must be **provable**. This is, unfortunately, very hard to achieve. In order to better illustrate this point, in Section 3.6.1 we present a coalition-related weakness in a recently proposed group signature scheme.

The second problem is more subtle. It is related to the notoriously difficult task of credential revocation in regular (non-group) public-key signature schemes. However, as discussed below, revocation in the context of group signatures is a more complicated problem.

3.5 Member Deletion

One intuitive and seemingly effective way to delete group members is via Certificate Revocation Lists (CRLs). A group manager periodically issues a new CRL wherein each entry identifies a group member (or a specific group certificate) whose membership has been revoked. There are many ways to generate and distribute CRLs. Unfortunately, the CRL approach is unsuitable for group signatures for the following reasons:

[3]For instance, electronic lotteries.

- Since group signatures are anonymous and unlinkable, how can the group manager "identify" a deleted member?

- If the group manager reveals certain (hereto secret) values corresponding to a deleted member, can the anonymity and unlinkability of **past signatures** of the said member be preserved?

The last item is probably the most important issue in member deletion. The preservation of security of past signatures after deletion is reminiscent of Perfect Forward Secrecy (PFS), a property often sought in key distribution protocols. If a member leaves the group voluntarily or a member is excluded for some administrative reason (such as a job promotion) there is nothing to gain (and much to loose) in sacrificing the security of past signatures.

However, in some cases, PFS is not desirable. For instance, suppose that a malicious member of a stock brokerage firm has been detected. It is surely beneficial – in the process of deletion – to inform all potential verifiers (clients) who might hold old signatures by the crooked broker.

A more interesting scenario is the compromise of a member's secrets. Suppose that a member's group certificate and/or group private key are inadvertently divulged. The group manager faces a real problem: to reveal this member's secrets means that all of its past signatures can be linked! Since there is no reason to doubt the validity of past signatures (the deleted member was not malicious) sacrificing unlinkability is clearly undesirable.

Another member deletion tactic is to change the group public-key. After generating the new group public-key, the group manager simply issues a new set of membership certificates to all remaining members while the rest are automatically excluded. This is an acceptable solution for small and stable groups. In large groups (e.g., lawyers in the state of California) this approach is unworkable due to frequent membership changes (e.g., lawyers being disbar-ed), the computational expense of re-issuing membership certificates and change notifications for all remaining members. Also, we assumed thus far that a member's new certificate can be issued unilaterally (off-line) by the group manager. However, some schemes may require interaction as part of repeated certificate issuance. (Initial certificate issuance always requires interaction). This complicates member deletion even further.

In summary, we have only *scratched the surface* of the member deletion problem. Since it appears that no proposed group signature scheme adequately addresses it, efficient and secure member deletion remains a pressing and interesting open problem.

3.6 Difficulty of Coalition-Resistance

A coalition attack occurs when a subgroup of group members pool their secrets together in order to produce a perfectly valid group signature. Such a signature is *untraceable*, i.e, the `OPEN` algorithm will fail to reveal the signer's identity.

In this section, we discuss a group signature scheme susceptible to coalition attacks. The aim is to gain a better understanding of how a coalition attack happens and, in the long run, to develop coalition-resistant techniques. Our discussion also illustrates that coalition-resistance is certainly one of the most challenging properties in designing a secure group signature scheme.

Following the definition in Section 3.2, each scheme discussed below is presented as a set of five algorithms: `SETUP`, `JOIN`, `SIGN`, `VERIFY` and `OPEN`. We are mainly concerned with the `JOIN` algorithm which enables a new user to join the group. As a result, the new user is issued a *membership certificate* which is then used to generate group signatures. We show how the knowledge of one or several certificates enables a coalition to derive a new (and valid) membership certificate, thus resulting in untraceable group signatures.

3.6.1 Camenisch-Stadler Scheme

Many ground-breaking cryptographic protocols are based on new number theoretic assumptions. There are two ways to invalidate these assumptions. One is by proving that they are false, the other – by proving that they are unsuitable for a specific protocol. The latter is the case with one of the assumptions underlying the Camenisch and Stadler's group signature scheme [37]. We briefly present some attacks on [37] and then propose some simple ways of preventing them.

Remark 3.6.1 *The following text describes two attack scenarios. It is necessary to stress that these are not all-out attacks that really* break *the scheme. Instead, we refer to them as* quasi-attacks *(for lack of better term) that exploit or break only one specific assumption.*

A *coalition* attack occurs when a subset of group members pool together their secrets and generate perfectly valid group signatures. Such signatures have the property that even the group manager is unable to open them. The coalition-resistance of the (basic) Camenisch-Stadler group signature relies on the following assumption (see Assumption 5.1 in [32, pp. 99–100] and [37, Section 4]).

Assumption 3.6.2 (Camenisch-Stadler) *Let n be an RSA-modulus, $e \in \mathbb{Z}^*_{\varphi(n)}$, and $a \in \mathbb{Z}^*_n$ an element of large order such that computing discrete logarithms w.r.t. base a is* *infeasible. We assume that it is hard to compute a pair (x, v) of integers satisfying:*

$$v^e \equiv 1 + a^x \pmod{n} \tag{3.1}$$

if the factorization of n is unknown. Furthermore, we assume that this is true even when other pairs (x_i, v_i) satisfying the above equation are known.

Loosely speaking, given one or more group certificate of the form $(a^x + 1)^d$, where $d \cdot e \equiv 1$ mod $\varphi(n)$, it should be hard to generate a new valid group certificate without the aid of the group manager.

However, consider the following special cases. Let $(a^x + 1)^d$ be a valid group certificate. It can be re-written as $(a^x + 1)^d = a^{xd}(1 + a^{-x})^d$. Now, suppose that $x = ke$ for some integer k, so that $a^{xd}(1 + a^{-x})^d = a^k(1 + a^{-x})^d$. Obviously, multiplying it by the inverse of a^k mod n we get the new certificate $(1 + a^{-x})^d$. This attack allows a *single* group member to produce a valid and untraceable signature. Hence, the scheme does not satisfy the *traceability* property. [4]

This attack can be trivially prevented, for example, by requiring the group member to prove that x is relatively prime with e (i.e. gcd(x,e)=1) without revealing x itself. However, as we now demonstrate, it is possible to generate a valid group certificate even when x is a generic value, e.g., if gcd(x,e)=1.

1. The first member registers and obtains a certificate A=$(a^x + 1)^d = a^{xd}(1 + a^{-x})^d$.
2. The second member then registers a^{-x} and obtains $B = (a^{-x} + 1)^d$ as its certificate.
3. They compute the inverse of B modulo n (using Euler's algorithm) and multiply it by A. The end-result is that two colluding members obtain the value $a^{xd} = A \cdot B^{-1}$ $= a^{xd}(1 + a^{-x})^d \cdot (1 + a^{-x})^{-d}$ mod n. got a^{xd}, quite awkward.

   Remark 3.6.3 *Possession of this value is dangerous: colluding members can take different strategies to generate a new group certificate. For instance, the value a^{xd} can be exponentiated with a random c, obtaining a^{cxd} and its inverse a^{-cxd} mod n.*

4. Then, a third group member registers a^{cx} and obtains $(a^{cx} + 1)^d$ as its certificate. Multiplying $(a^{cx} + 1)^d$ by a^{-cxd} yields a new and untraceable certificate $(a^{-cx} + 1)^d$ (in fact, $(a^{cx} + 1)^d \cdot a^{-cxd} = (a^{cx-cx} + a^{-cx})^d)$. It is also possible to generalize this strategy.

[4] Notice that the group member is required to show knowledge of the a's exponent in $(1+a^{-x})^d$ by means of a standard challenge-response protocol. Even though, the group member does not know $-x$ mod $\varphi(n)$, since $\varphi(n)$ is secret, this task can be easily accomplished by changing the sign of cx in the response, where c here represents the challenge. This relative "problem" also affects the Schnorr and Poupard-Stern signature schemes when the signer works in groups of unknown order.

A simple way to prevent this kind of attack is to substitute 1 in the certificate structure $(a^x + 1)^d$ with a random value t resulting in $(a^x + t)^d$ as the new certificate form.[5] Thus, Assumption 3.6.2 should be modified as follows.

Assumption 3.6.4 *Let n be an RSA-modulus, $e \in \mathbb{Z}^*_{\varphi(n)}$, and $a \in \mathbb{Z}^*_n$ an element of large order such that computing discrete logarithms w.r.t. base a is infeasible. We assume that it is hard to compute a pair (x, v) of integers satisfying:*

$$v^e \equiv t + a^x \pmod{n} \tag{3.2}$$

if the factorization of n is unknown and t is selected randomly. Furthermore, we assume that this is true even when other pairs (x_i, v_i) satisfying the above equation are known.

The scheme must be modified slightly in order to work with this new certificate form but the extra cost is a single exponentiation. Another simple fix, not mutually exclusive with the above, is to randomize (or "salt") the new member's membership key (and certificate) as part of JOIN, for example by selecting α, β and releasing the certificate $(a^{\alpha x+\beta} + t)^d$.

However, we cannot prove that the proposed fixes result in a secure form of membership certificates, i.e., that coalition attacks are, in fact, infeasible. We only showed that 1, as a redundancy factor, is a wrong choice for the CS97 scheme.

However, we would like to propose a new challenge with this new certificate's form, i.e. $(a^x+t)^d$ where t is a random value and x is the group member's secret exponent randomized by the group manager.

We stress that t must be chosen randomly, in particular, the discrete logarithm (base a) of t must be unknown. In fact, if the constant 1 in Eq. (3.1) is replaced by a constant t such that discrete logarithm of t (base a) is known, it is equally easy to derive a new certificate. Suppose that $t \equiv a^c \pmod{n}$ and c is known. Then, replacing 1 by t, Equation (3.1) becomes

$$v^e \equiv t + a^x \equiv a^{x-c}(a^{2c-x} + a^c) \equiv (a^{(x-c)d})^e(t + a^{2c-x}) \pmod{n}\ . \tag{3.3}$$

Hence if (x_1, v_1) is a valid pair, also is $(2c - x_1, v_1/a^{(x_1-c)d} \bmod n)$. Assuming that $\gcd(x_1 - c, e) = 1$ and, given two pairs: (x_1, v_1) and (x_2, v_2) with $x_2 = 2c - x_1$, we can compute $T \equiv (v_1/v_2)^\alpha a^\beta \equiv a^{(\alpha(x_1-c)+\beta e)d} \equiv a^d \pmod{n}$ where α and β are given by the extended Euclidean algorithm, i.e., $\alpha(x_1 - c) + \beta e = 1$. Therefore, using a a third pair (x_3, v_3), a new pair $(x, v) = (2c - x_3, v_3/T^{(x_3-c)} \bmod n)$ can be derived.

[5] This has been also suggested by J. Camenisch (via private communication).

3.7 Towards Coalition-Resistance

The previous sections make it fairly evident that finding a group certificate resistant against coalition attacks is not straightforward. As further evidence, we have been involved in a lengthy process of creating and subsequently breaking many certificate constructs and the underlying number-theoretic assumptions. As a research report and a learning exercise based on our efforts, we now describe some attempts to find a coalition-resistant group certificate. We conclude by presenting a candidate certificate structure based on a standard assumption widely believed to be hard.

Our starting point is the group signature proposed in [15]. In [15], it was noted that in certain group signature settings (we refer to as *coalition-oblivious*) group members have no incentives to collude. One example application is the *digital lottery*, where each group member anonymously signs its own numbered ticket (to make it valid) and the group manager later distributes prizes to the right holders of the randomly drawn ticket numbers.

For coalition-oblivious settings, [15] presents a viable alternative to costly group signatures that satisfies all the security properties in Section 3.2, except coalition-resistance.[6]

We now review the certificate used in [15] and then describe one of the (failed) attempts to make it coalition resistant without losing in efficiency.[7]

Let n be product of two safe primes p, q, i.e., $p = 2p' + 1$ and $q = 2q' + 1$ with p', q' appropriately selected primes. Let v, d be elements such that v is prime and $vd \equiv 1 \pmod{\varphi(n)}$. Let a be a generator of a subgroup $G \subset \mathbb{Z}_n^*$ of order $p'q'$, namely G is the set of quadratic residues modulo n and $a = \bar{a}^2 \bmod n$, for a randomly chosen $\bar{a} \in \mathbb{Z}_n^*$ (with overwhelming probability the order of a is $p'q'$; nevertheless, one may test whether $\gcd(a \pm 1, n) = 1$)). The certificate is $A(x) = (a^x\, a_0)^d \bmod n$ where x is the group member's secret (randomized by the group manager) with $0 < x < v$, and $a_0 \in G$ is a constant such that $\log_a a_0$ is unknown. The protocol is run as follows ([15]):

1. Using El Gamal encryption, the signer encrypts $A(x)$ with the group manager's public-key $z = a^s \bmod n$.
2. Then, he proves to the verifier that he indeed encrypted his valid certificate, i.e., given $\alpha = A(x)\, z^r \bmod n$ and $\beta = a^r \bmod n$, the verifier computes $\gamma = \alpha^v / a_0 \bmod n$

[6] We note that finding a signature scheme that does not satisfy any other property is quite straightforward. For instance, a group signature not satisfying the *unlinkability* property can be built with just certified pseudonyms.

[7] We are grateful to Jacques Traore for pointing out, via private communication, weaknesses in a early (fortunately unpublished) proposal of coalition-resistant group signature. These weaknesses were independently discovered by Marc Joye.

and checks that γ is a *representation* w.r.t. bases a and z, where the a's exponent is in $]0, v[$ and the z's exponent equals $v \log_a \beta$.

All these checks can be done via standard and efficient techniques.

Evidently, the certificate is not coalition-resistant: two colluding members can compute a new and untraceable certificate as follows. Assuming $A(x_1)$, $A(x_2)$ are the certificates of the two colluding members, it is sufficient to compute $[A(x_1)]^2/A(x_2) \bmod n$ which yields $A(2x_1 - x_2)$. It is likely the case that $0 < 2x_1 - x_2 < v$. More importantly, two colluding members can easily compute $a^d \bmod n$, and, thus, generate arbitrary certificates.

To make $A(x)$ coalition-resistant we considered methods forcing the signer to prove some non-linear relations of his secret x. Hence, we came up with the following certificate structure:

$$A(x) = ({a_2}^{x^2}\, {a_1}^{x}\, a_0)^d \bmod n \tag{3.4}$$

with a_0, a_1, a_2 random quadratic residues of order $p'q'$. First of all, notice that x must be in $]0, v[$ since it is easy to compute $A(x + tv)$ from $A(x)$ for an integer t. Hence, the signer should prove that $0 < x < v$ and that, given $A(x)^v/a_0$, the exponent of a_2 is the square of the exponent of a_1 (via well-known techniques). However, we now show that this certificate, although seemingly *robust*, is not coalition-resistant.

Define $J(x_1, x_2) :\equiv A(x_1)/A(x_2) \equiv {a_2}^{d(x_1^2 - x_2^2)}\, {a_1}^{d(x_1 - x_2)} \pmod n$. So, given a third certificate $A(x_3)$ and assuming w.l.o.g. that $\gcd(x_1 - x_2, x_1 - x_3) = 1$, we have

$$J(x_1, x_2)^\alpha\, J(x_1, x_3)^\beta \equiv {a_2}^{d[\alpha(x_1^2 - x_2^2) + \beta(x_1^2 - x_3^2)]}\, {a_1}^d \pmod n \tag{3.5}$$

where α, β are given by the extended Euclidean algorithm, i.e., $\alpha(x_1 - x_2) + \beta(x_1 - x_3) = 1$. Assume furthermore that we are given a fourth certificate $A(x_4)$ such that $\gcd(x_1 - x_2, x_1 - x_4) = 1$. Similarly, the extended Euclidean algorithm yields $\eta, \xi \in \mathbb{Z}$ s.t. $J(x_1, x_2)^\eta\, J(x_1, x_4)^\xi \equiv {a_2}^{d[\eta(x_1^2 - x_2^2) + \xi(x_1^2 - x_4^2)]}\, {a_1}^d \pmod n$. Hence, letting $\tau := [\alpha(x_1^2 - x_2^2) + \beta(x_1^2 - x_3^2)] - [\eta(x_1^2 - x_2^2) + \xi(x_1^2 - x_4^2)]$, it follows that

$$K :\equiv \frac{J(x_1, x_2)^\alpha\, J(x_1, x_3)^\beta}{J(x_1, x_2)^\eta\, J(x_1, x_4)^\xi} \equiv {a_2}^{d\tau} \pmod n\ . \tag{3.6}$$

Moreover, noting that v is prime, $\gcd(v, \tau)$ will very likely be 1. Therefore, there exist $\rho, \sigma \in \mathbb{Z}$ s.t. $\rho v + \sigma\tau = 1$ and thus

$${a_2}^d \equiv {a_2}^{d(\rho v + \sigma\tau)} \equiv {a_2}^\rho\, K^\sigma \pmod n\ . \tag{3.7}$$

Once ${a_2}^d$ is known, ${a_1}^d$ is given, for example, by Eq. (3.5) as

$${a_1}^d \equiv \frac{J(x_1, x_2)^\alpha\, J(x_1, x_3)^\beta}{({a_2}^d)^{\alpha(x_1^2 - x_2^2) + \beta(x_1^2 - x_3^2)}} \pmod n, \tag{3.8}$$

and, hence, ${a_0}^d$ is given from any certificate $A(x_i)$ ($i \in \{1,2,3,4\}$) as

$$ {a_0}^d \equiv \frac{A(x_i)}{({a_2}^d)^{x_i^2}\,({a_1}^d)^{x_i}} \pmod n \ . \tag{3.9} $$

In summary, from their respective certificates $A(x_i)$, four colluding group members are able to derive the values of ${a_2}^d, {a_1}^d, {a_0}^d \pmod n$, and, thus, compute a new valid membership certificate $A(x) = ({a_2}^d)^{x^2}\,({a_1}^d)^x\,{a_0}^d \bmod n$ whatever the value of x.

3.7.1 Strong RSA-based Scheme

It is clear that this approach is generally flawed and one must re-examine the problem and look for other solutions. In particular, an ideal solution would have its security based on one or more *well-known* number-theoretic problems.

This was first addressed successfully in a recent work by Camenisch and Michels [33]. The security of their scheme is based on the strong RSA assumption that we will formally define later (see Assumption 3.8.2). [8]

We now briefly describe the Camenisch-Michels scheme and then show how its efficiency can be improved [12].

Let n be product of safe primes and $G = \langle g \rangle$ be a cyclic subgroup of $\mathbb{Z}_n^*$ such that the Jacobi symbol $(g|n) = 1$. The group manager randomly selects $z, h \in G$ with the same order and publishes (n, g, z, h) and a prior determined range R. These parameters must be publicly verifiable. During the `JOIN` operation, the group member selects secret primes $e \in R$, $\bar{e}$ and sends $\tilde{e} = e\bar{e}$, $\tilde{z} = z^{\bar{e}} \bmod n$ to the group manager, along with evidences that he knows $\log_z \tilde{z}$, that e is indeed in R, and, finally, that $\tilde{e}$ is product of two primes. Notice that the latter is a quite expensive proof. Later, the group manager computes the value $u = \tilde{z}^{1/\tilde{e}} \bmod n$, which is equivalent to $u = z^{1/e} \bmod n$. The group certificate is the pair (u, e). Notice that the group manager does not know e that thus becomes the group member's secret.[9] In order to `SIGN`, the group member encrypts, via El Gamal, the value u with the group manager's public key $y = g^x \bmod n$, i.e. $\alpha = u\,y^w \bmod n$, $\beta = g^w \bmod n$, commits e by computing $\gamma = g^e h^w$, and, finally, shows knowledge of two values s_1, s_2 such that $\alpha^{s_1}/y^{s_2} \pmod n$ equals z and that $\log_{y^e}(\alpha^e/z) = \log_g \beta$. All without revealing the secret e. This is done via standard non-interactive techniques based on proofs of knowledge.

Let us remark that these techniques may not work properly into the group G suggested in the original scheme [33]. This is because the group member must check that an element

[8] Intuitively, the strong RSA assumption states that given a random $z \in \mathbb{Z}_n^*$ with n an RSA modulus, finding a pair $(u, e) \in \mathbb{Z}_n^* \times \mathbb{Z}$ such that $e > 1$ and $u^e = z$ is hard to solve.

[9] Although $z^{e+k/e}$, $k \in \mathbb{Z}$ can be easily derived from $z^{\bar{e}}$ and $z^{\tilde{e}}$, but this does not seem to hurt.

$a \in G$, $a \neq \pm 1$ is of large order by testing whether $\gcd(a \pm 1, n) = 1$. Unfortunately, this implies only that the order of a is at least $p'q'$, specifically $\mathrm{ord}_G(a) = p'q'$ or $2p'q'$. For the sake of simplicity, we consider the proof of equality of two discrete logarithms, i.e. given $y_1^{x_1}, y_2^{x_2} \pmod n$, proving that $x_1 = x_2$. But if $\mathrm{ord}_G(y_1) = \mathrm{ord}_G(y_2) = 2p'q'$, it might be the case that $x_1 \equiv x_2 \pmod{p'q'}$ but $x_1 \not\equiv x_2 \pmod 2$. A better choice for G is the set of quadratic residues modulo n, denoted by Q_n. In fact, notice that, when n is product of safe primes, Q_n is a cyclic group of order $p'q'$. In this case, either the group manager shows evidences that the public values, like z, are quadratic residues or the group member (and the verifier) modifies the non-interactive proofs by squaring the parameters involved.

3.8 Our Proposal

Finding a more efficient scheme than that in [33] is a difficult task. We can do so only by simplifying the `JOIN` operation. More precisely, we can avoid the proof that an element is product of two primes (as already mentioned, this is quite an expensive technique). We anticipate that the new scheme is a bit more expensive than that in [15] (however it satisfies the coalition-resistance property) yet more efficient than that in [33] since the `JOIN` operation requires fewer exponentiations.

The basic idea is to extend the certificate structure in [15], i.e., $A(x) = (a^x\, a_0)^d \bmod n$. As already mentioned, n is the product of two safe primes, a is a generator of Q_n (the set of quadratic residues), and $a_0 \in Q_n$ is a constant such that $\log_a a_0$ is unknown. Now, during the `JOIN` operation the group member P_i receives the certificate $A(x) = (a^{x_i}\, a_0)^{d_i} \bmod n$, where x_i is the group member's secret (randomized by the group manager) with $\lambda_1 < x_i < \lambda_2$. That is, the group manager randomly selects a prime e_i and computes d_i such that $e_i\, d_i \equiv 1 \pmod{\varphi(n)}$ for each group member P_i, with $0 < \lambda_1 < \lambda_2 < e_i$ (λ_1, λ_2 are fixed *a priori*). We explicitly require that for any $i \neq j$, $e_i \neq e_j$ must hold. The security of this certificate against any coalition attacks is based on the strong RSA assumption. The main difference with the certificate structure in [33] is that, now, the primes e_i are exclusively selected by the group manager, thus avoiding any further costly checks.

We observe that many group signature schemes (e.g., [33, 36, 37]) can be viewed as making use of two different ordinary signature schemes: one to generate membership certificates as part of `JOIN` and another to actually generate group signatures as part of `SIGN` (cf. [36]). Consequently, the properties of any secure group signature scheme must include the Unforgeability property for each of the two ordinary signature schemes. It is easy to see that each of: Traceability and Exculpability, map into the Unforgeability property for the two respective signature schemes. Furthermore, together they ensure that a group-signature scheme is unforgeable, i.e., that only group members are able to sign.

Many recent group signature schemes involve applying two types of non-group signature schemes: for issuing certificates and for actual group-signatures, respectively. The security of the former, in particular, is of immediate relevance since it assures, among other things, the coalition-resistance property of a group signature scheme. The reasoning for this assertion is fairly intuitive:

> Each group member obtains a unique certificate from the group manager as part of `JOIN` where each certificate is actually a signature over a secret message chosen by each member. As a coalition, all group members can be collectively thought of as a single adversary mounting an adaptive chosen message attack consisting of a polynomially many instances of `JOIN`.

As an illustration, we consider scheme of Camenisch and Michels [33, 34] which is the only group signature scheme heretofore shown secure (also under the Strong RSA Assumption). Recall the certificate format in [33, 34]: (u, e) where $u = \tilde{z}^{\tilde{e}^{-1}}$, $\tilde{z} = z^{\hat{e}}$, $\tilde{e} = \hat{e} * e$ and z is a public base. The values e (a prime!) and $\hat{e}$ are chosen by the joining member while u is computed by the group manager. One simply rewrite the certificate as: $(z^{e^{-1}}, e)$.

This is, in fact, a special case (the only one fully proven secure) of a signature in the Gennaro et al. scheme [56]. The interesting fact is that the work of Camenisch and Michels actually precedes the that of Gennaro et al. However, we point out that the seemingly small step between coalition-resistance in group signatures and resistance to adaptive chosen text attacks in plain signatures was apparently (and unfortunately) not spotted in [33, 34].

The art in designing a group signature scheme is to find a signature scheme for certification of membership such that the second signature scheme, i.e., the one that members use to sign on the group's behalf becomes efficient. Typically, this second scheme is derived (using the Fiat-Shamir heuristic) from a proof of knowledge of a membership certificate. Hence, the certification signature scheme must be such that this latter proof is efficiently realizable. Since proving knowledge of a pre-image under a hash-function is in general not efficiently possible, signature schemes that can be used must allow to to replace the hash-function with a suitable other function. What is helping in finding a replacement for the hash-function is the fact that `JOIN` is an interactive protocol and hence the choice of what gets signed can be influenced by the signer/group-manager.

3.8.1 Preliminaries

This section reviews some cryptographic assumptions and introduces the building blocks necessary in the subsequent design of our group signature scheme. (It can be skipped with no significant loss of continuity.)

3.8.1.1 Number-Theoretic Assumptions

The *Strong RSA Assumption* was independently introduced by Barić and Pfitzmann [17] and by Fujisaki and Okamoto [54]. It strengthens the widely accepted RSA Assumption that finding e^{th}-roots modulo n – where e is the public, and thus fixed, exponent – is hard to the assumption that finding an e^{th}-root modulo n for any $e > 1$ is hard. We give hereafter a more formal definition.

Definition 3.8.1 (Strong RSA Problem) *Let $n = pq$ be an RSA-like modulus and let G be a cyclic subgroup of $\mathbb{Z}_n^*$ of order $\#G$, $|\#G|_2 = \ell_G$. Given n and $z \in G$, the* Strong RSA Problem *consists of finding $u \in G$ and $e \in \mathbb{Z}_{>1}$ satisfying $z \equiv u^e \pmod{n}$.*

Assumption 3.8.2 (Strong RSA Assumption) *There exists a probabilistic poly-time algorithm $\mathcal{K}$ which on input 1^{ℓ_G} outputs a pair (n, z) such that, for all probabilistic poly-time algorithms $\mathcal{P}$, the probability that $\mathcal{P}$ can solve the Strong RSA Problem is negligible.*

The *Diffie-Hellman Assumption* [49] appears in two "flavors": (i) the Computational Diffie-Hellman Assumption and (ii) the Decisional Diffie-Hellman Assumption. For a thorough discussion on the subject, we refer the reader to [24]. We formalize these notions below although we do not utilize them until Section 3.8.3.

Definition 3.8.3 (Decisional Diffie-Hellman Problem) *Let $G = \langle g \rangle$ be a cyclic group generated by g, of order $\#G$, $|\#G|_2 = \ell_G$. Given g, g^u, g^v, and $g^w \in G$, the* Decisional Diffie-Hellman Problem *consists of deciding whether the elements g^{uv} and g^w are equal.*

This problem gives rise to the *Decisional Diffie-Hellman Assumption*, which was first explicitly mentioned in [26] by Brands although it was already implicitly assumed in earlier cryptographic schemes.

Assumption 3.8.4 (Decisional Diffie-Hellman Assumption) *There exists a probabilistic poly-time algorithm $\mathcal{K}$ which on input 1^{ℓ_G} outputs a 4-tuple (g, g^u, g^v, g^w) that is computationally indistinguishable from a 4-tuple randomly chosen from G^4.*

We stress that the Decisional Diffie-Hellman Problem is easier than the *(Computational) Diffie-Hellman Problem* which consists of finding g^{uv} from g^u and g^v; therefore, the Decisional Diffie-Hellman Assumption is a *stronger* assumption.

If n is a safe RSA-modulus (i.e., $n = pq$ with $p = 2p' + 1$, $q = 2q' + 1$, and p, q, p', q' all prime), it is more cautious to work in the subgroup of quadratic residues modulo n, that is, in the cyclic subgroup $\mathrm{QR}(n)$ generated by an element of order $p'q'$, since DDH holds and the order of the group has not small factors.

The next corollary shows that it is easy to find a generator g of $\mathrm{QR}(n)$: it suffices to choose an element $a \in \mathbb{Z}_n^*$ satisfying $\gcd(a \pm 1, n) = 1$ and then to take $g = a^2 \bmod n$. We then have $\mathrm{QR}(n) = \langle g \rangle$. (By convention, $\gcd(0, n) := n$.)

Proposition 1 *Let $n = pq$, where $p \neq q$, $p = 2p' + 1$, $q = 2q' + 1$, and p, q, p', q' are all prime. The order of the elements in $\mathbb{Z}_n^*$ are one of the set $\{1, 2, p', q', 2p', 2q', p'q', 2p'q'\}$. Moreover, the order of $a \in \mathbb{Z}_n^*$ is equal to $p'q'$ or $2p'q'$ $\iff$ $\gcd(a \pm 1, n) = 1$.* □

Corollary 3.8.5 *Let n given as in Proposition 1. Then, for any $a \in \mathbb{Z}_n^*$ s.t. $\gcd(a \pm 1, n) = 1$, $\langle a^2 \rangle \subset \mathbb{Z}_n^*$ is a cyclic subgroup of order $p'q'$.* □

Remark 3.8.6 *Notice that 4 $(= 2^2)$ always generates $\mathrm{QR}(n)$ whatever the value of a safe RSA modulus n. Notice also that the Jacobi symbol $(g|n) = +1$ does* not *necessarily imply that g is a quadratic residue modulo n but merely that $(g|p) = (g|q) = \pm 1$, where $(g|p)$ (resp. $(g|q)$) denotes the Legendre symbol*[10] *of g modulo p (resp. q). For example, $(2|55) = (2|5)\,(2|11) = (-1)\,(-1) = +1$; however, there is no integer x such that $x^2 \equiv 2 \pmod{55}$.*

3.8.1.2 Signatures of Knowledge

So-called zero-knowledge proofs of knowledge allow a prover to demonstrate the knowledge of a secret w.r.t. some public information such that no other information is revealed in the process. The protocols we use in the following are all three-move protocols and are can be proven zero-knowledge against an honest verifier. These kind of protocol can be performed non-interactively making use of an ideal hash-function $\mathcal{H}$ (*à la* Fiat-Shamir [52]). According to [37], we will call thus obtained schemes *signatures of knowledge*. An example is the Schnorr signature schemes [98] which can be seen as being derived that way from a proof of knowledge of the discrete logarithm of the signer's public key.

In the following, we consider three building blocks: signatures of knowledge of (i) a discrete logarithm; (ii) two discrete logarithms that are equal; and (iii) a discrete logarithm lying in a given interval. (Other building blocks can be defined in the same way (see [9]); they can also be combined naturally (see [32]).) All these blocks are constructed over a cyclic group $G = \langle g \rangle$ whose order $\#G$ is unknown; however its bit-length ℓ_G (i.e., the integer ℓ_G s.t. $2^{\ell_G - 1} \leq \#G < 2^{\ell_G}$) is publicly known. Fujisaki and Okamota [54] show that under the SRSA, the standard proof of knowledge protocols that work for a group of known order are also proofs of knowledge in this setting. We define the *discrete logarithm* of $y \in G$ w.r.t. base g as any integer $x \in \mathbb{Z}$ such that $y = g^x$ in G. We note $x = \mathrm{Dlog}_g\, y$. We assume

[10] By definition, the Legendre symbol $(g|p) = +1$ if g is a quadratic residue modulo p, and -1 otherwise.

a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ which maps a binary string of arbitrary length to a k-bit hash value. We also assume a security parameter $\epsilon > 1$.

Showing the knowledge of the discrete logarithm of $y = g^x$ can be done easily as stated by the following definition (cf. [98]).

Definition 3.8.7 *A pair* $(c, s) \in \{0,1\}^k \times \pm\{0,1\}^{\epsilon(\ell_G+k)+1}$ *verifying* $c = \mathcal{H}(m\|y\|g\|g^s\, y^c)$ *is a signature of knowledge of the discrete logarithm of* $y = g^x$ *w.r.t. base* g*, on a message* $m \in \{0,1\}^*$.

The party in possession of the secret $x = \mathrm{Dlog}_g\, y$ is able to compute the signature by choosing a random $t \in \pm\{0,1\}^{\epsilon(\ell_G+k)}$ and then computing c and s as:

$$c = \mathcal{H}(m\|y\|g\|g^t), \ s = t - cx \quad (\text{in } \mathbb{Z}) \ .$$

A slight modification of the previous definition enables to show the knowledge and equality of two discrete logarithm of, say y_1 and y_2 w.r.t. bases g and h, i.e., knowledge of an integer x satisfying $y_1 = g^x$ and $y_2 = h^x$.

Definition 3.8.8 *A pair* $(c, s) \in \{0,1\}^k \times \pm\{0,1\}^{\epsilon(\ell_G+k)+1}$ *verifying* $c = \mathcal{H}(m\|y_1\|y_2\|g\|h \|g^s\, {y_1}^c\|h^s\, {y_2}^c)$ *is a signature of knowledge of the discrete logarithm of both* $y_1 = g^x$ *w.r.t. base* g *and* $y_2 = h^x$ *w.r.t. base* h*, on a message* $m \in \{0,1\}^*$.

The party in possession of the secret x is able to compute the signature, provided that $x = \mathrm{Dlog}_g\, y_1 = \mathrm{Dlog}_h\, y_2$, by choosing a random $t \in \pm\{0,1\}^{\epsilon(\ell_G+k)}$ and then computing c and s as:

$$c = \mathcal{H}(m\|y_1\|y_2\|g\|h\|g^t\|h^t), \ s = t - cx \quad (\text{in } \mathbb{Z}) \ .$$

In Definition 3.8.7, a party shows the knowledge of the discrete logarithm of y w.r.t. base g. The order of g being unknown, this means that this party knows an integer x satisfying $y = g^x$. This latter condition may be completed in the sense that the party knows a discrete logarithm x lying in a given interval. It is a slight modification of a protocol appearing in [55].

Definition 3.8.9 *A pair* $(c, s) \in \{0,1\}^k \times \pm\{0,1\}^{\epsilon(\ell+k)+1}$ *verifying* $c = \mathcal{H}(m\|y\|g\|g^{s-cX}\, y^c)$ *is a signature of knowledge of the discrete logarithm of* $y = g^x$ *w.r.t. base* g *and that this logarithm lies in* $]X - 2^{\epsilon(\ell+k)}, X + 2^{\epsilon(\ell+k)}[$*, on a message* $m \in \{0,1\}^*$.

From the knowledge of $x = \mathrm{Dlog}_g\, y \in \,]X - 2^\ell, X + 2^\ell[$, this signature is obtained by choosing a random $t \in \pm\{0,1\}^{\epsilon(\ell+k)}$ and computing c and s as:

$$c = \mathcal{H}(m\|y\|g\|g^t), \ s = t - c(x - X) \quad (\text{in } \mathbb{Z}) \ .$$

Remark 3.8.10 *Note that, although the party knows a secret x in $]X - 2^\ell, X + 2^\ell[$, the signature only guarantees that x lies in the extended interval $]X - 2^{\epsilon(\ell+k)}, X + 2^{\epsilon(\ell+k)}[$.*

The security of all the presented building blocks has been proven in the random oracle model [20] under the Strong RSA Assumption in [34, 54, 55]. That is, if $\epsilon > 1$, then the corresponding interactive protocols are statistical (honest-verifier) zero-knowledge proofs of knowledge.

3.8.2 The New Signature Scheme

This section describes a new public key digital signature scheme with the security (i.e, unfogeability property) based on the Strong RSA Assumption.

3.8.2.1 The Model

Definition 3.8.11 *A* digital signature scheme *consists of three algorithms*

`GEN`: *On input of a security parameter 1^ℓ this probabilistic algorithm output the signer's secret and public keys, x and y, respectively.*

`SIG`: *On input of the the signer's secret and public keys and a message $m \in \{0,1\}^*$ this algorithm outputs a signature s on m.*

`VER`: *On input of a message m, a signature σ, and the public key y_s of a signer, the the algorithm `VER` outputs `true` or `false`*

A signature scheme must satisfy the following properties:

Correctness: *Signatures produced by the signer with `SIG` must be accepted by `VER`.*

Unforgeability [58]: *A signature scheme must be existentially unforgeable under an chosen message attack. That is, we require that every attacker has a negligible probability of success in the following game. The attacker is allowed to sequentially obtain signature on polynomially many messages of his choosing (i.e., messages are allowed to depend on signatures that the adversary has seen). He is then required to produce as output a message m for which he did not receive a signature and a second string s. If `VER`$(m, s, y) =$ `true` then the attacker is successful and, hence, the signature scheme is vulnerable to* existential forgery.

3.8.2.2 The Scheme

Let $\epsilon > 1$ be a security parameter and let ℓ_p, $\ell_{\lambda_1} > \ell_{\lambda_2}$, $\ell_{\gamma_1} > \ell_{\gamma_2}$ denote lengths. Define the integral ranges $\Lambda =]2^{\ell_{\lambda_1}} - 2^{\ell_{\lambda_2}}, 2^{\ell_{\lambda_1}} + 2^{\ell_{\lambda_2}}[$ and $\Gamma =]2^{\ell_{\gamma_1}} - 2^{\ell_{\gamma_2}}, 2^{\ell_{\gamma_1}} + 2^{\ell_{\gamma_2}}[$ such that for all $(x, e) \in \Lambda \times \Gamma$, we have $0 < x + 2^{2\ell_p} < e$. Finally, let $\{\mathcal{H}\}$ be a collision-resistant hash function family such that, for all $\mathcal{H} \in \{\mathcal{H}\}$, $\mathcal{H} : \{0,1\}^* \to \Lambda$.

To generate his public and secret key, a signer runs the following algorithm.

<u>`GEN:`</u>

1. Select random secret ℓ_p-bit primes p', q' such that both $p = 2p'+1$ and $q = 2q'+1$ are also prime. Set the modulus $n = pq$.
2. Choose two random elements $a, a_0 \in_R \mathrm{QR}(n)$ (of order $p'q'$).
3. Choose a hash-function $\mathcal{H} \in \{\mathcal{H}\}$
4. The public key consists of the tuple $(n, a, a_0, \mathcal{H})$.
5. The corresponding secret key consists of (p', q')

To sign a message $m \in \{0,1\}^*$ the signer uses the following algorithm.

<u>`SIG:`</u>

1. Choose a prime $e \in_R \Gamma$ that was not used before.
2. Choose a random integer $r \in_R \Lambda$
3. Compute $x := \mathcal{H}(m \| e \| r)$.
4. Compute $u := (a^x a_0)^{1/e} \bmod n$
5. Output the signature (u, e, r)

In Step 1 of `SIG`, there are two possibilities for choosing e. It can either be chosen randomly, or the signer stores the prime last used and just takes the next larger prime.

Checking whether a tuple (u, e, r) is a valid signature on a message $m \in \{0,1\}^*$ with respect to the public key n can be done as follows.

<u>`VER:`</u>

1. Check whether $(u, e, r) \in \mathbb{Z}_n^* \times \Gamma \times \Lambda$.
2. Compute $\tilde{x} := \mathcal{H}(m \| e \| r)$.
3. Check whether $u^e \equiv a^{\tilde{x}} a_0 \pmod{n}$.
4. Output the signature `true` if none of the checks failed.

Note the verifier does not need not to check that e is a prime.

3.8.2.3 Efficiency Analysis

The cost of the SIG algorithm can be broken down into three components:

1. generation of a random (or unused) prime e from the range of Γ
2. computation of its inverse, e^{-1}
3. computation of u which requires 2 exponentiations: one full, with e^{-1}, and one "small", with x (since $\mathcal{H}() \in \Lambda$)

The cost of the last step can be reduced by amending GEN to generate a_0 as a power of a, i.e., choose $\hat{r} \in_R \Lambda$ and compute $a_0 = a^{\hat{r}}$. (Of course, $\hat{r}$ would then become part of the secret key). This amendment allows us to avoid the "small" exponentiation in the last step above, i.e., the signer would perform only one (full) exponentiation with the exponent $(x + \hat{r})e^{-1}$.

The only possible drawback is the potential loss in the range of a_0 since it is no longer generated independently from a. However, we note that a similar speedup was proposed by Cramer and Shoup [46] where it was claimed that, since, a (h in [46]) is highly likely a generator of $\mathrm{QR}(n)$, the distribution of the resultant public key does not change significantly.

The cost of signature verification in our scheme is dominated by Step 3 (see above) which requires two exponentiations: one full to compute u^e, and one "small" to compute $a^{\tilde{x}}$. However, the verification equation can be changed to $u^e(a^{-1})^{\tilde{x}} \equiv a_0 \pmod n$ and hence the computation gets reduced to about one full exponentiation.

3.8.2.4 Security Analysis

This section shows that our signature scheme indeed satisfies the requirement for a secure signature scheme according to Definition 3.8.11. The correctness property follows from inspection of the scheme. It remains to prove the schemes security against an adaptively chosen message attack. Similar to [56] we require that:

1. for every $\mathcal{H} \in \{\mathcal{H}\}$, all primes $e \in \Gamma$, and every two messages m_1 and m_2 the distribution $\mathcal{H}(m_1\|e\|r)$ and $\mathcal{H}(m_1\|e\|r)$ induced be the random choice of r are statistically close
2. the Strong RSA Assumption holds in a world where there exists an oracle that – on input a message m, a prime $e \in \Gamma$, and an $x \in \Lambda$ – outputs an $r \in \Lambda$ such that $x = \mathcal{H}(m\|e\|r)$.

Note that the latter assumption is not restricting as is shown in [56].

Theorem 3.8.12 *The signature scheme presented above is secure against adaptively chosen messages attack under the Strong RSA Assumption and the further assumption that there exists a family of hash functions $\{\mathcal{H}\}$ satisfying the above requirements.*

Proof. Assume that the attacker $\mathcal{A}$ queries signature for K messages and then outputs a signature $(\hat{u}, \hat{e})$ on the message $\hat{m}$. We now show that if we take control over the hash-function, then we can use this attacker to break the Strong RSA Assumption, i.e., we are given a z and an n and must find an w and v such that $w^v \equiv z \pmod{n}$.

Let $((u_1, e_1, r_1), m_1), \ldots, ((u_K, e_K, r_K), m_K)$ denote the signature-message pairs that are constructed during the interaction with $\mathcal{A}$. In order for $\mathcal{A}$ to be successful its output $((\hat{u}, \hat{e}, \hat{r}), \hat{m})$ must satisfy $(\hat{u}, \hat{e}) \neq (u_i, e_i)$ for $1 \leq i \leq K$. Depending of whether $e_i \not| \hat{e}$ for some i, there are two games to calculate a pair $(w, v) \in \mathbb{Z}_n^* \times \mathbb{Z}_{>1}$ satisfying $w^v \equiv z \pmod{n}$ from which we randomly chose one each time then play with the attacker. As mentioned before, we are assuming that there is an oracle that input a message m, a prime $e \in \Gamma$, and an $x \in \Lambda$ outputs an $r \in \Lambda$ such that $x = \mathcal{H}(m\|e\|r)$. The adversary is allowed to query this oracle as well. The first of the two game goes as follows:

1. Select $x_1, \ldots, x_K \in \Lambda$ and $e_1, \ldots, e_K \in \Gamma$.
2. Set $a = z^{\prod_{1 \leq l \leq K} e_l} \bmod n$.
3. Choose $r \in_R \{0,1\}^{2\ell_p}$ and set $a_0 = a^r \bmod n$.
4. For all $1 \leq i \leq K$, compute $u_i = z^{(x_i + r)\prod_{1 \leq l \leq K; l \neq i} e_l} \bmod n$.
5. Start $\mathcal{A}$, feed it the (u_i, e_i, r_i) (where we get r_i from the oracle) and eventually obtain $(\hat{x}; [\hat{u} = (a^{\hat{x}} a_0)^{1/\hat{e}} \bmod n, \hat{e}], \hat{r})$ with $\hat{x}, \hat{r} \in \Lambda$ and $\hat{e} \in \Gamma$.
6. If $\gcd(\hat{e}, e_j) \neq 1$ for all $1 \leq j \leq K$ output `fail` and stop. Otherwise, let $\tilde{e} := (\hat{x} + r)\prod_{1 \leq l \leq K} e_l$. (Note that $\hat{u}^{\hat{e}} \equiv z^{\tilde{e}} \pmod{n}$.) Since $\gcd(\hat{e}, e_j) = 1$ for all $1 \leq j \leq K$, we have $\gcd(\hat{e}, \tilde{e}) = \gcd(\hat{e}, (\hat{x}+r))$. Hence, by the extended Euclidean algorithm, there exist $\alpha, \beta \in \mathbb{Z}$ s.t. $\alpha\,\hat{e} + \beta\,\tilde{e} = \gcd(\hat{e}, (\hat{x} + r))$. Therefore, letting $w := z^{\alpha}\,\hat{u}^{\beta} \bmod n$ and $v := \hat{e}/\gcd(\hat{e}, (\hat{x} + r)) > 1$ since $\hat{e} > (\hat{x} + r)$, we have $w^v \equiv z \pmod{n}$.

The previous game is only successful if $\mathcal{A}$ returns a new signature with $\gcd(\hat{e}, e_j) = 1$ for all $1 \leq j \leq K$. We now present a game that solves the Strong RSA Problem in the other case, that is, when $\gcd(\hat{e}, e_j) \neq 1$ for some $1 \leq j \leq K$. (Note that $\gcd(\hat{e}, e_j) \neq 1$ means $\gcd(\hat{e}, e_j) = e_j$ since e_j is prime.)

1. Select $x_1, \ldots, x_K \in \Lambda$ and $e_1, \ldots, e_K \in \Gamma$.
2. Choose $j \in_R \{1, \ldots, K\}$ and set $a = z^{\prod_{1 \leq l \leq K; l \neq j} e_l} \bmod n$.
3. Choose $r \in_R \{0,1\}^{2\ell_p}$ and set $u_j = a^r \bmod n$ and $a_0 = u_j^{e_j}/a^{x_j} \bmod b$.
4. For all $1 \leq i \leq K$ $i \neq j$, compute $u_i = z^{(x_i + e_j r - x_j)\prod_{1 \leq l \leq K; l \neq i,j} e_l} \bmod n$.

5. Start $\mathcal{A}$, feed it the (u_i, e_i, r_i) (where we get r_i from the oracle) and eventually obtain $(\hat{x}; [\hat{u} = (a^{\hat{x}}\, a_0)^{1/\hat{e}} \bmod n, \hat{e}, \hat{r}])$ with $\hat{x}, \hat{r} \in \Lambda$ and $\hat{e} \in \Gamma$.
6. If $\gcd(\hat{e}, e_j) \neq e_j$ output `fail` and stop. Otherwise, we have $\hat{e} = t\, e_j$ for some t and can define $Z := \hat{u}^t / u_j \bmod n$ if $\hat{x} \geq x_j$ and $Z := u_j / \hat{u}^t \bmod n$ otherwise. Hence, $Z \equiv (a^{|\hat{x} - x_j|})^{1/e_j} \equiv (z^{|\tilde{e}|})^{1/e_j} \pmod{n}$ with $\tilde{e} := (\hat{x} - x_j) \prod_{1 \leq l \leq K; l \neq j} e_l$. Since $\gcd(e_j, \prod_{\substack{1 \leq l \leq K \\ l \neq j}} e_l) = 1$, it follows that $\gcd(e_j, |\tilde{e}|) = \gcd(e_j, |\hat{x} - x_j|)$. Hence, there exist $\alpha, \beta \in \mathbb{Z}$ s.t. $\alpha\, e_j + \beta\, |\tilde{e}| = \gcd(e_j, |\hat{x} - x_j|)$. Therefore, letting $u := z^{\alpha}\, Z^{\beta} \bmod n$ and $e := e_j / \gcd(e_j, |\hat{x} - x_j|) > 1$ since $e_j > |\hat{x} - x_j|$, we have $u^e \equiv z \pmod{n}$.

Consequently, by playing randomly one of Game 1 or 2 with $\mathcal{A}$ one can solve the Strong RSA Problem. Since the latter is assumed to be infeasible, it follows that no such attacker can exist. □

3.8.2.5 Related Work

We now compare the present scheme with some recent results.

The scheme due to Gennaro et al. [56] is simpler and seemingly more efficient than our scheme. The scheme is simpler since it appears as a true hash-and-sign scheme very close to RSA. It uses a similar variation of the Strong RSA Assumption for the proof of security as we do. However, their requirements for a suitable hash-function are non-standard, e.g., it is required to be division-intractable.

An interesting sidenote is that the only practical realization of a suitable hash function presented in [56] is the so-called *chameleon hashing* which outputs primes. This yields a signature scheme that requires (as part of hashing the message) the signer to generate a random-looking prime. The cost of signing thus becomes roughly the same as in our present scheme: generation of a large prime, computation of its inverse and a single exponentiation. The cost of verification is one exponentiation plus the cost of a message hash which is quite expensive due to the special hash-function used.

Comparing our scheme to the one by Cramer/Shoup, we find that are similar in many aspects of security properties and associated costs.

The public key size in our scheme is somewhat smaller than its counterpart in Cramer/Shoup. The latter consists of a tuple (n, h, x, e') where n is a modulus, h and x are elements of QR_n, and e' is a prime. In contrast, our scheme's public key is a tuple where n is a modulus, a and a_0 are elements of QR_n, and h is a hash function which is, incidentally, also needed in a Cramer/Shoup public key. Thus, the size difference is due to the prime e' in the latter. A Cramer/Shoup signature is a tuple (y, y', e) where e is a (small) prime, $y' \in QR_n$, and $y \in Z_n^*$, i.e., both are n-bit integers. This is about the same as for our scheme. The cost of signing in [46] amounts to generating a prime, computing its inverse and three exponentiations of which two are small (each with an exponent from the range of the underlying hash function) and one is full. Hence, the cost of signing is somewhat higher in [46] than in our scheme. Signature verification in the Cramer/Shoup translates into two small exponentiations (with two bases each) which is a bit more efficient than in our scheme (one full exponentiation).

3.8.3 The New Group-Signature Scheme

In this section we construct a new group signature scheme building upon the results attained thus far. In particular, as described in detail below, the group membership certificate signing function is derived from the digital signature proposed in the previous section.

3.8.3.1 The Scheme

Let $\epsilon > 1$ be a security parameter and let ℓ_p, $\ell_{\lambda_1} > \ell_{\lambda_2}$, $\ell_{\gamma_1} > \ell_{\gamma_2}$ denote lengths. Define the integral ranges $\Lambda =]2^{\ell_{\lambda_1}} - 2^{\ell_{\lambda_2}}, 2^{\ell_{\lambda_1}} + 2^{\ell_{\lambda_2}}[$ and $\Gamma =]2^{\ell_{\gamma_1}} - 2^{\ell_{\gamma_2}}, 2^{\ell_{\gamma_1}} + 2^{\ell_{\gamma_2}}[$ such that for all $(x, e) \in \Lambda \times \Gamma$, we have $0 < x + 2^{2\ell_p} < e$. Finally, let $\mathcal{H}$ be a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$.

The initial phase involves the group manager (*GRMGR*) setting the group public and secret keys: Y and S.

SETUP :

1. Select random secret ℓ_p-bit primes p', q' such that $p = 2p' + 1$ and $q = 2q' + 1$ are prime. Set the modulus $n = pq$.
2. Choose random elements $a, a_0, g, h \in_R \mathrm{QR}(n)$ (of order $p'q'$).
3. Choose a random secret element $x \in_R \mathbb{Z}^*_{p'q'}$ and set $y = g^x \bmod n$.
4. The group public key is: $\mathcal{Y} = (n, a, a_0, y, g, h)$.
5. The corresponding secret key (known only to *GRMGR*) is: $\mathcal{S} = (p', q', x)$.

Remark 3.8.13 *The group public key Y is made available via the usual means (i.e., embedded in some form of a public key certificate signed by a trusted authority). We note that, in practice, components of Y must be verifiable to prevent framing attacks. In particular, Corollary 3.8.5 provides an efficient way to prove that an element has order $p'q'$ in $\mathrm{QR}(n)$. GRMGR also needs to provide a proof of n being the product of two safe primes p, q with $p = 2p' + 1$ and $q = 2q' + 1$ (see [35]).*

Suppose now that a user wants to join the group. We assume that communication between the user and the group manager is secure, i.e., private and authentic. The selection of per-user parameters is done as follows:

<u>`JOIN:`</u>

1. User P_i generates a secret exponent $\hat{x}_i \in_R]0, 2^{\lambda_2}[$, a random integer $\hat{r} \in_R]0, 2^{2\ell_p}[$ and sends $C_1 = g^{\hat{x}_i} h^{\hat{r}} \bmod n$ to *GRMGR* and proves *GRMGR* the knowledge of the representation of C_1 w.r.t. g and h.
2. *GRMGR* selects α_i and $\beta_i \in_R]0, 2^{\lambda_2}[$ at random and sends (α_i, β_i) to P_i.
3. User P_i computes $x_i = 2^{\lambda_1} + (\alpha_i \hat{x}_i + \beta_i \bmod 2^{\lambda_2})$ and sends *GRMGR* the value $C_2 = a^{x_i} \bmod n$. The user also proves *GRMGR* that (1) the discrete log of C_2 to the base a lies in Λ and (2) knowledge of integers u, v, and w such that (i) u lies in $]-2^{\lambda_2}, 2^{\lambda_2}[$ (ii) u equals the discrete log of $C_2/a^{2^{\lambda_1}}$ to the base a (iii) $C_1^{\alpha_i} g^{\beta_i}$ equals $g^u (g^{2^{\lambda_2}})^v h^w$ (see Definition 3.8.9). (The statements (i–iii) prove that the users secret $x_i = \log_a C_2$ is correctly computed from C_1, α_i, and β_i.)
4. If all the proofs are correct *GRMGR* selects a random prime $e_i \in_R \Gamma$ and computes $A_i := (C_2\, a_0)^{1/e_i} \bmod n$. Finally, *GRMGR* sends P_i the new membership certificate $[A_i, e_i]$. (Note that $A_i = (a^{x_i}\, a_0)^{1/e_i} \bmod n$.)
5. User P_i verifies that $a^{x_i} a_0 \equiv {A_i}^{e_i} \pmod{n}$.

Remark 3.8.14 *As part of* `JOIN`*, GRMGR creates a new entry in the membership table and stores* $\{[A_i, e_i]$, `JOIN` *transcript*$\}$ *in the new entry. (*`JOIN` *transcript is formed by the messages received from the user in the steps above. It is assumed to be signed by the user with some form of a long-term credential.)*

Armed with a membership certificate $[A_i, e_i]$, a group member can generate anonymous and unlinkable group signatures on a generic message m:

SIGN:

1. Generate a random value $w \in_R \{0,1\}^{2\ell_p}$ and compute:

$$T_1 = A_i\, y^w \bmod n, \quad T_2 = g^w \bmod n, \quad T_3 = g^{e_i}\, h^w \bmod n \ .$$

2. Randomly choose $r_1 \in_R \pm\{0,1\}^{\epsilon(\ell_{\gamma_2}+k)}$, $r_2 \in_R \pm\{0,1\}^{\epsilon(\ell_{\lambda_2}+k)}$, $r_3 \in_R \pm\{0,1\}^{\epsilon(\ell_{\gamma_1}+2\ell_p+k+1)}$ and $r_4 \in_R \pm\{0,1\}^{\epsilon(2\ell_p+k)}$, and compute:

   (a) $d_1 = {T_1}^{r_1}/(a^{r_2}\, y^{r_3}) \bmod n$, $d_2 = {T_2}^{r_1}/g^{r_3} \bmod n$, $d_3 = g^{r_4} \bmod n$, $d_4 = g^{r_1}\, h^{r_4} \bmod n$;
   (b) $c = \mathcal{H}(m\|g\|h\|y\|a_0\|a\|T_1\|T_2\|T_3\|d_1\|d_2\|d_3\|d_4)$;
   (c) $s_1 = r_1 - c(e_i - 2^{\ell_{\gamma_1}})$, $s_2 = r_2 - c(x_i - 2^{\ell_{\lambda_1}})$, $s_3 = r_3 - c\, e_i\, w$, $s_4 = r_4 - c\, w \quad (\text{in } \mathbb{Z})$.

3. Send: $\{c, s_1, s_2, s_3, s_4, T_1, T_2, T_3\}$ to verifier.

Verifier checks the validity of the signature of the message m as follows:

VERIFY:

1. Compute:

$$c' = \mathcal{H}(m\|g\|h\|y\|a_0\|a\|T_1\|T_2\|T_3\|{a_0}^c\, {T_1}^{s_1 - c2^{\ell_{\gamma_1}}}/(a^{s_2 - c2^{\ell_{\lambda_1}}}\, y^{s_3}) \bmod n\| \\ {T_2}^{s_1 - c2^{\ell_{\gamma_1}}}/g^{s_3} \bmod n\|{T_2}^c\, g^{s_4} \bmod n\|{T_3}^c\, g^{s_1 - c2^{\ell_{\gamma_1}}}\, h^{s_4} \bmod n) \ .$$

2. Accept the signature if only if $c = c'$ and $s_1 \in \pm\{0,1\}^{\epsilon(\ell_{\gamma_2}+k)+1}$, $s_2 \in \pm\{0,1\}^{\epsilon(\ell_{\lambda_2}+k)+1}$, $s_3 \in \pm\{0,1\}^{\epsilon(\ell_{\lambda_1}+2\ell_p+k+1)+1}$, $s_4 \in \pm\{0,1\}^{\epsilon(2\ell_p+k)+1}$.

In the event that the actual signer must be subsequently identified (e.g., in case of a dispute) *GRMGR* executes the following procedure:

OPEN:

1. Check signature validity via `VERIFY` procedure.
2. Recover A_i (and thus the identity of P_i) as $A_i = T_1/{T_2}^x \bmod n$.
3. Prove that $\mathrm{Dlog}_g\, y = \mathrm{Dlog}_{T_2}[T_1/A_i \bmod n]$ (see Definition 3.8.8).

3.8.3.2 Security of the Proposed Scheme

In this section we assess the security of the new group signature scheme. We first need to prove that the following theorems hold.

Theorem 3.8.15 *Under the Strong RSA Assumption, a group certificate* $[A_i = (a^{x_i}\, a_0)^{1/e_i} \bmod n, e_i]$ *with* $x_i \in \Lambda$ *and* $e_i \in \Gamma$ *can be generated only by the group manager.*

Proof. The proof of this theorem is quite similar to the proof of Theorem 3.8.12. The difference is that we do not need the "hash-function" oracle to assure that the x_i are those that we prepared (in fact, there is no hash-function involved in here). Instead, here we can use the proof of knowledge of the representation of C_1 w.r.t. g and h that users have to do in the first step of the `JOIN` protocol to extract the value $\hat{x}_i$ and then set α_i and β_i such that the resulting x_i will be equal to the one we prepared. ∎

Theorem 3.8.16 *Under the Strong RSA Assumption, the interactive protocol underlying the group signature scheme is a statistical zero-knowledge (honest-verifier) proof of knowledge of a group certificate.*

Proof. The proof that the interactive protocol is statistical zero-knowledge is quite standard (e.g., see [32]). We restrict our attention the proof of knowledge part.

We have to show that the knowledge extractor is able to recover the group certificate once it has found two accepting tuples. Let $(T_1, T_2, T_3, d_1, d_2, d_3, d_4, c, s_1, s_2, s_3, s_4)$ and $(T_1, T_2, T_3, d_1, d_2, d_3, d_4, \tilde{c}, \tilde{s}_1, \tilde{s}_2, \tilde{s}_3, \tilde{s}_4)$ be two accepting tuples. Since $d_3 \equiv g^{s_4}\, {T_2}^{c} \equiv g^{\tilde{s}_4}\, {T_2}^{\tilde{c}} \pmod n$, it follows that $g^{s_4 - \tilde{s}_4} \equiv {T_2}^{\tilde{c}-c} \pmod n$. Letting $\delta_4 = \gcd(s_4 - \tilde{s}_4, \tilde{c} - c)$, by the extended Euclidean algorithm, there exist $\alpha_4, \beta_4 \in \mathbb{Z}$ s.t. $\alpha_4\,(s_4 - \tilde{s}_4) + \beta_4\,(\tilde{c} - c) = \delta_4$. Hence,

$$g \equiv g^{(\alpha_4\,(s_4 - \tilde{s}_4) + \beta_4\,(\tilde{c}-c))/\delta_4} \equiv ({T_2}^{\alpha_4}\, g^{\beta_4})^{\frac{\tilde{c}-c}{\delta_4}} \pmod n \ .$$

Note that we cannot have $\tilde{c} - c < \delta_4$ since otherwise ${T_2}^{\alpha_4}\, g^{\beta_4}$ is a $(\frac{\tilde{c}-c}{\delta_4})^{\text{th}}$ root of g, which contradicts the Strong RSA Assumption. Thus, we have $\tilde{c} - c = \delta_4 = \gcd(s_4 - \tilde{s}_4, \tilde{c} - c)$; or equivalently, there exists $\tau_4 \in \mathbb{Z}$ s.t. $s_4 - \tilde{s}_4 = \tau_4(\tilde{c} - c)$. So, since $s_4 + c\,w = \tilde{s}_4 + \tilde{c}\,w$, we have $\tau_4 = w$ and thus obtain

$$A_i = \frac{T_1}{y^{\tau_4}} \bmod n \ .$$

Moreover, since $d_4 \equiv g^{s_1}\, h^{s_4}\, (T_3\, g^{-2^{\ell_{\gamma_1}}})^{c} \equiv g^{\tilde{s}_1}\, h^{\tilde{s}_4}\, (T_3\, g^{-2^{\ell_{\gamma_1}}})^{\tilde{c}} \pmod n$, we have $g^{s_1 - \tilde{s}_1} \equiv (T_3\, g^{-2^{\ell_{\gamma_1}}})^{\tilde{c}-c}\, h^{\tilde{s}_4 - s_4} \equiv (T_3\, g^{-2^{\ell_{\gamma_1}}}\, h^{-\tau_4})^{\tilde{c}-c} \pmod n$. Let $\delta_1 = \gcd(s_1 - \tilde{s}_1, \tilde{c} - c)$. By the extended Euclidean algorithm, there exist $\alpha_1, \beta_1 \in \mathbb{Z}$ s.t. $\alpha_1\,(s_1 - \tilde{s}_1) + \beta_1\,(\tilde{c} - c) = \delta_1$. Therefore, $g \equiv g^{(\alpha_1\,(s_1 - \tilde{s}_1) + \beta_1\,(\tilde{c}-c))/\delta_1} \equiv [(T_3\, g^{-2^{\ell_{\gamma_1}}}\, h^{-\tau_4})^{\alpha_1}\, g^{\beta_1}]^{\frac{\tilde{c}-c}{\delta_1}} \pmod n$. This, in turn, implies by the Strong RSA Assumption that $\tilde{c} - c = \delta_1 = \gcd(s_1 - \tilde{s}_1, \tilde{c} - c)$; or equivalently that there exists $\tau_1 \in \mathbb{Z}$ s.t. $s_1 - \tilde{s}_1 = \tau_1(\tilde{c} - c)$. Consequently, since $s_1 + c(e_i - 2^{\ell_{\gamma_1}}) = \tilde{s} + \tilde{c}(e_i - 2^{\ell_{\gamma_1}})$, we find

$$e_i = 2^{\ell_{\gamma_1}} + \tau_1 \ .$$

Likewise, from $d_2 \equiv {T_2}^{s_1} g^{-s_3} ({T_2}^{-2^{\ell_{\gamma_1}}})^c \equiv {T_2}^{\tilde{s}_1} g^{-\tilde{s}_3} ({T_2}^{-2^{\ell_{\gamma_1}}})^{\tilde{c}} \pmod{n}$, it follows that $g^{s_3 - \tilde{s}_3} \equiv ({T_2}^{\tau_1 + 2^{\ell_{\gamma_1}}})^{\tilde{c}-c} \pmod{n}$. Therefore, by the extended Euclidean algorithm, we can conclude that there exists $\tau_3 \in \mathbb{Z}$ s.t. $s_3 - \tilde{s}_3 = \tau_3(\tilde{c} - c)$. Finally, from

$$d_1 \equiv {T_1}^{s_1} a^{-s_2} y^{-s_3} ({T_1}^{-2^{\ell_{\gamma_1}}} a^{2^{\ell_{\lambda_1}}} a_0)^c \equiv {T_1}^{\tilde{s}_1} a^{-\tilde{s}_2} y^{-\tilde{s}_3} ({T_1}^{-2^{\ell_{\gamma_1}}} a^{2^{\ell_{\lambda_1}}} a_0)^{\tilde{c}} \pmod{n} ,$$

we get $a^{\tilde{s}_2 - s_2} \equiv ({T_1}^{\tau_1 + 2^{\ell_{\gamma_1}}} y^{-\tau_3} a^{-2^{\ell_{\lambda_1}}} {a_0}^{-1})^{\tilde{c}-c} \pmod{n}$ and similarly conclude that there exists $\tau_2 \in \mathbb{Z}$ s.t. $s_2 - \tilde{s}_2 = \tau_2(\tilde{c} - c)$. Therefore, since $s_2 + c(x_i - 2^{\ell_{\lambda_1}}) = \tilde{s}_2 + \tilde{c}(x_i - 2^{\ell_{\lambda_1}})$, we recover

$$x_i = 2^{\ell_{\lambda_1}} + \tau_2 ,$$

which concludes the proof. ∎

Corollary 3.8.17 *The `JOIN`-protocol is zero-knowledge w.r.t. the group manager and a^{x_i}. Furthermore, the user's secret key x_i is a random integer from Λ.*

The proof of the above corollary is straight forward.

Corollary 3.8.18 *In the random oracle model the group signature scheme presented in this section is secure under the Strong RSA Assumption.*

Proof. We have to show that our scheme satisfies all the properties listed in Definition 3.2.1.

Correctness: By inspection.

Anonymity: Given a signature, identifying the actual signer is computationally hard for everyone but the group manager: Because from Theorem 3.8.16 the underlying interactive protocol is statistically zero-knowledge, no information is statistically revealed by (c, s_1, s_2, s_3, s_4) in the random oracle model. So anonymity can only be revoked from (T_1, T_2, T_3). Since no one (but the group manager) knows which pair $[A_i, e_i]$ corresponds to which group member, anonymity is guaranteed.

Unlinkability: Deciding whether two different signatures were computed by the same group member is computationally hard: Let $(T_1, T_2, T_3, c, s_1, s_2, s_3, s_4)$ and $(\tilde{T}_1, \tilde{T}_2, \tilde{T}_3, \tilde{c}, \tilde{s}_1, \tilde{s}_2, \tilde{s}_3, \tilde{s}_4)$. As in the previous corollary, the problem of linking two signatures reduces to decide whether (T_1, T_2, T_3) and $(\tilde{T}_1, \tilde{T}_2, \tilde{T}_3)$ are linked. This can be done by looking either A_i or e_i is common to the two triples. This is however impossible under Decisional Diffie-Hellman Assumption.

Exculpability: Neither a group member nor the group manager can sign on behalf of other group members: First note that due to Corollary 3.8.17, the *GRMGR* does not get any information about a user's secret x_i apart from a^{x_i}. Thus, the value x_i is computationally hidden from *GRMGR*. Next note that T_1, T_2, and T_3 are an unconditionally binding commitments to A_i and e_i. One can show that, if the factorization of n would be publicly known, the interactive proof underlying the group signature scheme is a proof of knowledge of the discrete log of $A_i^{e_i}/a_0$ (provided that ℓ_p is larger than twice to output length of the hash function / size of the challenges). Hence, not even the group manager can sign on behalf of P_i since computing discrete logarithms is assumed to be infeasible.

Traceability: The group manager is able to open any valid group signature and *provably* identify the actual signer: Assuming that the signature is valid, this implies that T_1 and T_2 are of the required form and so A_i can be uniquely recovered. Due to Theorem 3.8.15 a group certificate $[A_i = A(x_i), e_i]$ with $x_i \in \Lambda$ and $e_i \in \Gamma$ can only be obtained from via the `JOIN`-protocol. Hence, the A_i recovered can be uniquely be linked to an instance of the `JOIN`-protocol and thus the user P_i who originated the signature can be identified.

Coalition-resistance: As mentioned earlier, this follows from traceability.

□

3.8.3.3 Comparisons

In terms of efficiency the existing group signature scheme can be divided into two classes: (I) those where the size of the group's public key and/or of signatures (linearly) depend on the number of group members and (II) those where it does not. The first schemes proposed all belong to the first class. Although most of those scheme are proven secure with respect to some standard cryptographic assumption such a the hardness of computing discrete logarithms, these are rather inefficient for larger groups. In the quest for the a scheme of class II a number of schemes have been proposed so far. With the exception of the recent Camenisch/Michels scheme [33, 34], all previously proposed group signature schemes of class II fall into two categories: 1) those that have been broken, and 2) those that have not been proven secure under standard cryptographic assumptions.

The only major difference between the Camenisch/Michels scheme and our new scheme is in the member-certificate format, i.e., the signature scheme used in the `JOIN` protocol. (All the other procedures of the two schemes have similar efficiency.) In [33, 34] – in order to provide security against an adaptive adversary – the new member and the group manager need to jointly generate a random prime without the group manager knowing this prime. This is done in a similar way as the choice of x_i in our protocol. However, since in [33, 34], the chosen random number must be a prime, that part of the protocol must repeated sufficiently many times to hit a prime. For reasonable security parameter this is about 600 times (compared to once in our scheme). Furthermore, when a prime is found, the new user must additionally prove the group manager that it is indeed prime. Thus our

JOIN-protocols is an order of magnitude more efficient.

3.9 Summary

This chapter presented a new digital signature scheme based on the strong RSA assumption and a very efficient and provably secure group signature scheme. Its performance (and apart from the one in [33]) also its security appears to significantly surpass those of prior art. Extending the scheme to a blind group-signature scheme or to split the group manager into a membership manager and a revocation manager is straight forward (cf. [33, 74]). Our scheme can also be turned into an identity escrow system [66], which is be seen as an anonymous group identification scheme with revocation. Such a scheme can be obtained just by using as identification protocol the interactive protocol underlying the SIGN procedure. The resulting scheme will be provable secure without assuming random oracles. An application of our ordinary signature scheme is "on-the-fly" generation of signature (cf. [57, 92]), where signing must be very fast, but one is allowed to use precomputed values: the prime e, $b = a^{1/e}$, and $b_0 = a_0^{1/e}$ could be precomputed and then signing a message m would only require to compute $x = \mathcal{H}(m\|e\|r)$ and $b^x b_0 \bmod n$, where x is small (e.g., 160 bit) and r some random number that could also be chosen during preprocessing.

Chapter 4

Authentication Services and Extensions in Group Setting

This chapter analyzes what could be achieved by simply adding further cryptographic techniques to the basic key agreement protocols described in chapter 2. It also discusses new concepts and authentication services unique to groups. Finally, it introduces some practical security services that can be constructed using any group signature scheme. Sample realizations of these services are provided. The results described in this chapter have appeared in [13, 14, 16].

4.1 Introduction

It is clear that the primary motivation for obtaining a group key is the ability to communicate securely and efficiently once a key is established. If all group members share a key, they can communicate using symmetric encryption. This is more efficient than schemes not requiring key establishment. It should also be clear that a group key agreement provides group implicit key authentication if each party of the group is assured that no other party outside the group itself may compute a particular secret key.

This chapter analyzes further cryptographic techniques, like key confirmation and entity authentication, and explores new concepts and services unique to groups.

4.2 Entity Authentication

As already mentioned, *key confirmation* (Def. 2.4.4 and [79]) is an important feature in key agreement protocols. Its purpose is to convince one or more parties that its peer (or a group of peers) is in possession of the key. It can be argued that key confirmation is not absolutely necessary if communication immediately follows key agreement, i.e., if a proper key is subsequently used for bi-directional data flows, key confirmation is achieved as a side-effect. However, in general, it is desirable to bundle key confirmation with key agreement for the following reasons:

1. it makes key agreement both a more robust and more autonomous operation
2. doing otherwise can lead to an incorrectly computed key not being detected later (since there may be a delay between key agreement and actual data communication)

On the other hand, it is not clear what key confirmation means in a peer group setting. Complete key confirmation (in the spirit of complete key authentication) would make it necessary for all group members to compute the key and then confirm to all other members the knowledge of the key. This would entail, at the very least, one round of n simultaneous broadcasts. We take a more practical approach by concentrating on key confirmation emanating from the group controller, the first group member to compute the actual key.

It turns out that the construction of A-GDH.2 (and SA-GDH.2) makes key confirmation fairly easy to add. The only change to both protocols is the addition to the last protocol message (the broadcast from M_n) of:

$$\alpha^{F(S_n(M_n))}$$

where $S_n(M_n)$ denotes the key as computed by M_n and $F()$ is as previously defined.

Upon receipt of the broadcast, each M_i computes its key $S_n(M_i)$ as before. Then, M_i verifies the computed key:

$$\alpha^{F(S_n(M_i))} \stackrel{?}{=} \alpha^{F(S_n(M_n))}$$

In both A-GDH.2 and SA-GDH.2, key confirmation coupled with implicit key authentication, has a nice side-effect of providing *entity authentication* of M_n to all other group members. Informally, this is because the upflow message in round i can be viewed as a random challenge (r_i being M_i's nonce) submitted to M_n (indirectly, through all other $M_j; j > i$). The last broadcast, then, can be viewed as M_n's reply to M_i's challenge encrypted under a secret key shared among M_i and M_n. To support our claim that the above results in entity authentication of M_n we need to show that M_n's reply is *fresh.* (That M_n's reply is *authentic* has been shown in Section 2.6.1.) Freshness, however, is evident from the way M_i computes the key: by exponentiating the value received from M_n with $(r_i \cdot\ K_{in}^{-1})$.

Remark 4.2.1 *In SA-GDH.2, for each M_i, key confirmation also results in* entity authentication *of all M_j, for $i < j \leq n$.*

4.3 Group Integrity and Verifiable Contributory

Including key confirmation in SA-GDH.2 leads us to an interesting observation:

> At the end of the protocol, each M_i knows that the key it holds, $S_n(M_i)$, has been contributed to by every group member.

This follows directly from the *complete group key authentication* property coupled with key confirmation. Recall that the former assures that, if any two distinct parties (M_i and M_j) share a key, that key must be contributed to by every group member. Adding key confirmation allows us to achieve a stronger goal: any group member can unilaterally establish that it is in possession of a correct key which has been contributed to by every member. This is both a novel and important feature of SA-GDH.2 and *a new security service* unique to group key agreement.

Definition 4.3.1 *(informal) A group key agreement protocol offers* **group integrity** *if each protocol party is assured of every other protocol party's participation in the protocol.*

Group integrity should not be confused with entity authentication. It is a *weaker* notion since group integrity does not guarantee freshness/timeliness. It only guarantees all parties' participation in the protocol and, likewise, all parties' awareness of the group membership.

Definition 4.3.2 *(informal) A group key agreement protocol is* **verifiable contributory** *if each protocol party is assured of every other protocol party's contribution to the group key.*

Note that *verifiable contributory* implies *group integrity* while the reverse is not true. For example, group integrity can be obtained by requiring every M_i to sign and forward (to all others) a statement certifying to its participation in the protocol. Also, verifiable contributory property does not imply that a group key is *not contributed to by an outside party*. As discussed in the section 4.4, an adversary can still inject some input into the group key.

4.4 The Elusive Key Integrity

Key integrity (Def. 2.4.5) is orthogonal to both key authentication and key confirmation. A key agreement protocol may offer one or both of the latter while at the same time not guaranteeing key integrity. Consider the following (3-party) SA-GDH.2 example:

Protocol SA-GDH.2 (example):
Round 1: M_1 selects random $r_1 \in_R \mathbb{Z}_q^*$.
$M_1 \longrightarrow M_2 : \alpha^{r_1 \cdot K_{12}}, \alpha^{r_1 \cdot K_{13}}$
Round 2: M_2 selects random $r_2 \in_R \mathbb{Z}_q^*$.
$M_2 \longrightarrow M_3 : \alpha^{r_1 \cdot K_{12}}, \alpha^{r_1 \cdot K_{13} \cdot r_2 \cdot K_{23}}, \alpha^{r_2 \cdot K_{21}}$
Round 3: M_3 selects random $r_3 \in_R \mathbb{Z}_q^*$, computes group key $S_3(3)$ and broadcasts:
$M_3 \longrightarrow M_1, M_2 : \alpha^{r_1 \cdot K_{12} \cdot K_{32} \cdot r_3}, \alpha^{r_2 \cdot K_{21} \cdot K_{31} \cdot r_3}, \alpha^{F(S_3(3))}$
M_2 computes $S_3(2)$, M_1 computes $S_3(1)$ and, finally, both M_1 and M_2 confirm the correctness of their respective keys against $\alpha^{F(S_3(3))}$.

This protocol offers complete group key authentication, key confirmation and, entity authentication of M_3. At the end, all parties wind up computing the same key. However, an adversary can exponentiate by a constant all values sent in round 1 (and/or round 2) and remain undetected. Suppose the adversary simply squares all values in round 2. Then, what M_3 actually receives is: $\alpha^{r_1 \cdot K_{12} \cdot 2}, \alpha^{r_1 \cdot K_{13} \cdot r_2 \cdot K_{23} \cdot 2}, \alpha^{r_2 \cdot K_{21} \cdot 2}$

As a result, M_3 computes $S_3(3) = \alpha^{r_1 \cdot r_2 \cdot r_3 \cdot 2}$ and both M_1 and M_2 compute the same value, i.e., the quadratic residue of the intended key. The key confirmation check does not help since the adversary introduces its input before M_n computes the group key.

We observe that, in SA-GDH.2, the adversary is only able to introduce multiplicative (in the exponent) input, i.e., it can cause the key to be K^C for some value C. The construction of the protocol precludes the adversary from introducing any other type of input, e.g., additive in the exponent.

This leads us to pose the following question:

> How important is key integrity in a *verifiable contributory* key agreement protocol?

In practice, we expect that key integrity can be easily assured via an external data integrity mechanism (e.g., SSL) used *hop-by-hop* in the upflow stage of the protocol. Consequently, if every protocol message between M_i and M_{i+1} in the i-th $(0 < i < n)$ protocol round is integrity-protected, the adversary is no longer able to introduce any "noise" into the group key. Note that, the last broadcast message does not need to be protected; any modification will be detected by the key confirmation check.

4.5 Other Security Services

As general rule, key establishment can be avoided as follows. A DPG member encrypts a message using a symmetric encryption function with a secret key K and then sends the cipher-text to the entire group along with $n-1$ versions of the key K; each encrypted using a public key of a member. Although this simple scheme has no (cryptographic) startup overhead, it is not contributory and becomes too expensive if the group is large or the volume of message traffic is high. Furthermore, it requires every member to be aware of the exact group membership at all times; something that can (if desired) be avoided with key agreement.

We believe that there are other incentives to consider. In particular, a shared group key can be used to provide a number of useful services (in an efficient manner). For example, we can use a secret group key (such as the one agreed upon in A-GDH.2) to derive a corresponding group Diffie-Hellman public key which can be subsequently embedded in a group certificate. This would allow any group member to use DSA [84] (or any ElGamal family) signatures to authenticate itself (as a group member) to both insiders and outsiders. The same group public-key can be viewed as long-term group Diffie-Hellman exponent and outsiders (including other groups) can establish shared keys with the entire group in a trivial manner.

4.5.1 Protocols

More in details, it is possible to achieve easily the following useful services:

- **Authentication to outsiders.** Suppose the group has agreed on a key K, thus it is sufficient to have $P(K) = \alpha^{F(K)}$ certified by some Certification Authority, where $F(\cdot)$ is an appropriate mapping function. Any group member would, then, be able to sign messages by using K as signing key and $P(K)$ as public key. Many signature algorithms work properly with the pair $(K, P(K))$, among them Schnorr and DSA. To authenticate a group member, an outsider could just send a random fresh message to him and asks to have it signed. This kind of authentication is completely anonymous, i.e., there is no way to establish who is the signer.

- **Intra-group authentication.** Having a group key K, it is possible to sign a generic message m just by computing $H = h_K(m)$, where $h_k(\cdot)$ is an appropriate MAC function $h_K(\cdot)$. The verifier (another group member), in possession of the secret K, can verify the signature on m by checking whether $h_K(m) = H$. If m is a random fresh message, it is then possible to authenticate the signer as a *current* group member.

- **Non-repudiation of group membership.** Since it is possible to sign as group member with $(K, P(K))$, thus the group membership cannot be repudiated.
- **Private communication within group.** With the group key K it is possible to encrypt messages, thus gaining confidentiality inside the group.
- **Private communication between outsiders and group.** The pair $(K, P(K))$ can be viewed as an ElGamal public key-pair; the public component thereof can be embedded in a group certificate. Outsiders can then use this key with ElGamal public-key encryption to communicate in secret with the entire group. Similarly, if we assume an outsider has a public-key as power of α, it is then possible to agree on a secret key via Diffie-Hellman and thus communicate privately.

4.6 New Directions in Group Signatures

In chapter 3 we discussed some outstanding issues that block wider acceptance and use of group signatures. We now switch gears and turn to "new and exciting" uses and extensions of group signatures.

4.6.1 Multi-Group Signatures

One novel and practical service that can be built using group signatures as a primitive is **multi-group signatures**. (Not to be confused with so-called multi-signatures [25].)

Informally, we define a multi-group signature (MGS) as a set of regular group signatures generated by the same entity (signer) who is a member of multiple groups. For example, a user belonging to two groups A and B may have a reason to sign something as a member of both groups. The need for such signatures may arise in real-life situations as illustrated in the following (admittedly somewhat contrived) examples:

1. A financial (e.g., loan) contract may need to be signed by an authorized loan officer of a bank and by a notary public. If the same person is a member of both groups (loan officers and notaries public) s/he can sign the document with a multi-group signature.
2. A republican senator (anonymously) releasing a statement to the media needs to assure the public that s/he is, in fact, a member of the senate and a republican party member. This can be achieved with a double-group signature.

3. An expert witness testifying in court (or in congress) who is afraid of retribution may need, nonetheless, to convince the public that s/he is, say, a medical doctor and an officer (of a certain rank) in the armed forces. Once again, a double-group signature is appropriate.

4. A person submitting a formal peer complaint to a state judicial review panel might have to sign the complaint text simultaneously as a: i) US citizen, ii) resident of that state and, iii) member of the bar. This requires a triple-group signature.

An interesting property required of MGS is the ability to **link** the individual group signatures comprising an MGS. In other words, it must be evident that the same signer produced each single-group signature which is part of an MGS.

At first, this might seem both counter-intuitive and counter-productive. Recall that one of the basic tenets of group signatures is **unlinkability**, i.e., the computational difficulty of establishing whether two distinct group signatures were produced by the same group member. However, we observe that the property we seek is **linkability** of group signatures **across different groups** rather than linkability of different signatures within a group. Thus, the desired property is not inherently in conflict with traditional group signatures.

We note that, in all other respects, an MGS scheme must have the same properties as a regular group signature scheme.

Now, whether an MGS scheme can be built out of existing group signature schemes is an entirely separate issue. We have considered several group signature schemes from the literature. Some are not easily converted, e.g., [31]. However, at least one scheme, CS97 [37], can be used as building block for a secure MGS scheme.

4.6.1.1 MGS Example

We begin with a short overview of the CS97 basic scheme, modified as shown in chapter 3, section 3.6.1.

The scheme is premised on two important building blocks: proofs (signatures) of knowledge of double discrete logarithms (SKLOGLOG) and signatures of knowledge of roots of discrete logarithms (SKROOTLOG). SKLOGLOG entails, given d, demonstrating knowledge of $log_b(log_a\ d)$. SKROOTLOG entails, given c, and e demonstrating knowledge of $(log_a c)^{1/e}$ where $(e, d = 1/e)$ is an RSA key-pair.

As part of setup, $GRMGR$ selects the following values:

- an RSA public-key parameter $n = pq$ for two large primes p, q and an RSA key-pair (e, d) $e \cdot\ d \equiv\ 1 \text{mod}\ n$.

- a cyclic group $G =< g >$ of order n where computing discrete logarithms is infeasible.
- an element $a \in \mathbb{Z}_n^*$ of large multiplicative order modulo both p and q.
- some other parameters not relevant to the ensuing discussion.

A prospective group member can join the group by selecting a random value x_1 randomized by $GRMGR$ resulting in the secret x (known only by the group member), computing $y = a^x \bmod n$ and $z = g^y$. Upon sending y to $GRMGR$ and proving knowledge of x, the member is issued a certificate: $(y+t)^d$.

A group signature over a message m is defined in CS97 as a tuple $< m,\ \bar{g},\ \bar{z},\ V_1,\ V_2 >$ where:

- m is the message to be signed
- $\bar{g} := g^r$ for $r \in_R \mathbb{Z}_n^*$
- $\bar{z} := \bar{g}^y$
- $V_1 :=$ SKLOGLOG$[x \ : \ \bar{z} = \bar{g}^{a^x}](m)$
- $V_2 :=$ SKROOTLOG$[v \ : \ \bar{z}\bar{g}^t = \bar{g}^{v^e}](m)$ for $v = (y+t)^d$

To verify a group signature, it is sufficient to check V_1 and V_2. We do not elaborate on other details/features of the scheme (such as the OPEN procedure) or its security properties as they are not germane to our discussion.

To further illustrate the MGS concept, we consider the specific case of double-group signatures; the technique can be trivially extended to n-group signatures. Suppose that the same group manager is used for both groups G_1 and G_2. (This is not an inherent requirement but it simplifies the example.) The setup procedure is almost the same as in CS97, i.e., $GRMGR$ picks n, $G =< g >$, a and other parameters as above. These values are common for both groups. However, $GRMGR$ selects different public-key-pairs: (e_1, e_1^{-1}) and (e_2, e_2^{-1}) for G_1 and G_2, respectively.

The JOIN procedure is largely the same as in CS97 except that a member who joins both groups (and is interested in computing MGSs) must use the same secret value x in both registrations[1]: $y_1 = y_2 = a^x \bmod n$. The two respective membership certificates are: $v_1 = (y+t)^{e_1^{-1}}$ and $v_2 = (y+t)^{e_2^{-1}}$.

To produce an MGS (a double-group signature, in this case) a member of both groups constructs a tuple $< m, SIG_1, SIG_2, V_3 >$ where:

[1]There does not appear to be any security risk in doing this.

- m is the message being signed.
- $SIG_1 =< \bar{g_1}, \bar{z_1}, V_{1,1}, V_{2,1} >$ is a single-group signature of message m where:
- $SIG_2 =< \bar{g_2}, \bar{z_2}, V_{1,2}, V_{2,2} >$ is also a single-group signature of m where:
- $V_3 = \text{SKEQLOG}(\bar{z_1}, \bar{g_1}, \bar{z_2}, \bar{g_2})$ is a signature of knowledge of equality of two discrete logarithms (of $\bar{z_1}$ and $\bar{z_2}$) with two different bases $\bar{g_1}$ and $\bar{g_2}$. More generally, SKEQLOG denotes a non-interactive proof of $log_{\bar{g_1}} \bar{z_1} = log_{\bar{g_2}} \bar{z_2}$. One efficient technique for constructing SKEQLOG is shown in chapter 3, section 3.8.1.2 (see also [42]).

As in CS97, $V_{1,1}$ and $V_{1,2}$ each demonstrate knowledge of the secret x, i.e., double-discrete logarithm of $\bar{z_1}$ and $\bar{z_2}$, respectively (which is x in both cases). $V_{2,1}$ and $V_{2,2}$ demonstrate knowledge of e_1-th (and e_2-th) roots of discrete logarithm of b_1 (and b_2), i.e., possession of membership certificates of groups G_1 and G_2. Finally, V_3 demonstrates equality of two discrete logarithms (z_1 and z_2) with two different bases (g_1 and g_2). The last step is crucial to "linking" the two signatures since it asserts that the same y is used in both signatures. Consequently, the same secret x is used in both signatures (since $y = a^x$.)

It appears that no reduction in security is incurred by using this MGS construct. (Assuming, of course, that the underlying CS97 basic scheme is itself secure.) In particular, the OPEN procedure remains the same and MGSs remain unlinkable much like their single-group counterparts.

A potential concern with the above extension is its relative inefficiency. since a generalized i-group MGS involves computing i regular single-group signatures and $(i-1)$ SKEQLOG-s. However, most expensive constructs are SKROOTLOGs and SKLOGLOGS required by CS97.

To conclude the MGS discussion, we need to point out that there is no compulsion to use the same group manager for MGS. In the CS97-based example above, the group public-key is:

$$\mathcal{Y} = [n, < g >, a, e, \text{other parameters}]$$

If two groups are managed separately, the two managers can independently compute their respective group public and private keys. A member registers with the same secret x in both groups and obtains two corresponding certificates. However, the problem arises when the member tries to prove that s/he uses the same x in generating SIG_1 and SIG_2. Since each group has a distinct base a, the member has to prove **equality of two double discrete logarithms**.

4.6.2 Sub-Group Signatures

Another interesting outgrowth of conventional group signatures is the notion of **sub-group signatures**. Informally, we define a sub-group signature (SGS) as a function of a member group $GR = \{M_1, ..., M_t\}$, a subset $S \subset\ GR$ and a message m as:

$$\mathcal{T} = \{SIG_j(m) | M_j \in S\} \ \ \#\mathcal{T} = i$$

Each $SIG_j(m)$ is a normal group signature by a group member M_j; for a given M_j, at most one SIG_j is in $\mathcal{T}$.

We should stress that the notion of SGS is very different from that of coalition group signatures as defined by Camenisch in [31]. Coalition group signatures are indistinguishable from single-member group signatures by verifiers. Only the group manager can (via OPEN) determine the coalition size and structure. In contrast, in SGS the size of the signing sub-group (or coalition) must be explicit and verifiable by anyone.

(As a sidenote, the scheme in [31] includes a method for obtaining coalition signatures; however, the coalition structures are fixed a priori by the group manager. This limits the utility of the whole concept.)

Consider the following example applications of SGS:

- A petition is circulated among members of a certain group. A number of members (i) sign the petition and then publicly announce that i members stand behind it. Any insider or outsider is able to verify that i distinct (and anonymous) members have indeed signed the petition.

- A military order must be signed by a least t ranked officers. Without exposing any internal command structure, t officers sign the order thus allowing anyone to verify its validity.

- A prospective club member must be vouched for by at least n existing members. These members must be distinct and, while n might be pre-set, there is no upper limit on the number of vouching members. The more members vouch, the faster the new member is admitted.

- A vote on a proposition is conducted by members of the board. Each member is given a single vote and vote tallies must be publicly verifiable.

It is also important to point out the differences between SGS and threshold schemes. Threshold schemes aim to partition a secret (e.g., a signature key) in a way that allows at least a pre-determined number (threshold) of shareholders to re-create a secret. In contrast, SGS does not involve a pre-determined threshold; any number of group members may sign a given message. The central goal of SGS is to demonstrate that a subset of a certain (unknown a priori) size has signed a given message.

In addition to the security properties of regular group signatures, a SGS scheme must feature:

- Compositional integrity: assure the verifier that all signatures comprising a SGS have been generated by distinct signers (group members).
- SGS unlinkability: multiple SGS-s generated by the same sub-group must be unlinkable. We note that this property does not necessarily follow from the unlinkability of the underlying group signature scheme (the latter only assures unlinkability of single-member signatures.)

SGS unlinkability is a natural property. Clearly, if single-member signatures are unlinkable, so should SGS-s. Compositional integrity, on the other hand, seems to contradict the unlinkability of regular group signatures. Recall that in Section 3.2 we defined **unlinkability** (in the context of regular group signatures) as the difficulty of deciding whether two distinct signatures were generated by the same group member. Indeed, we may need to weaken the unlinkability property in order to achieve compositional integrity. However, we emphasize that this should be done **only** for SGS; i.e., the structure of other types of group signatures (regular, multi-group) must remain unchanged.

The feasibility of SGS schemes is illustrated by a concrete example in the next section. Once again, we use the CS97 [37] group signature scheme as our main building block.

4.6.2.1 SGS Example

The actual SGS extension is quite trivial. The SETUP, JOIN and OPEN procedures are the same as in CS97. Only SIGN and VERIFY are modified slightly.

The modified SIGN procedure involves a new initial stage whereby the i multiple signers (sub-group members) agree on a common random, one-time base:

$$\bar{g} := g^r \text{ for } r \in_R \mathbb{Z}_n^*$$

In CS97, r (and, hence, $\bar{g}$) is generated at random by one signer. In our case, r needs to be generated collectively by the sub-group of signers. There are many methods for doing this; one possibility is to run a group Diffie-Hellman key agreement protocol (e.g., [29] or [102]) and, as a result, generate a common:

$$\bar{g} := g^r \text{ where } r = a^{r_1 \cdots r_i} (mod n)$$

and i is the size of the sub-group and each r_j is generated at random by the j-th signer.

Having agreed on a one-time base $\bar{g}$, each signer (M_j) computes a regular CS97 group signature: $SIG_j(m) =< m, \bar{g}, \bar{z}, V_{1,j}, V_{2,j} >$. Pooled together, the signatures form an SGS, $\mathcal{T} = \{SIG_j | 0 < j \leq i\}$. Of course, redundant m and $\bar{g}$ fields can be eliminated from $\mathcal{T}$.

The VERIFY procedure is also modified in accordance to the parameter i, the size of the (claimed) sub-group. The same $\bar{g}$ must be used as a base in verifying every signature in $\mathcal{T}$. In addition, for each $SIG_j(m)$ the verifier must check the uniqueness of $\bar{z}$ contained therein. (An occurrence of a duplicate $\bar{z}$ within a $\mathcal{T}$ indicates a redundant signature by a given signer.)

As a more concrete example, consider a SGS with a sub-group of size two:

$$< \ m, \ \bar{g}, \ SIG_1, \ SIG_2 \ > \quad \text{where:}$$

$$SIG_i =< \bar{z_i}, V_{1,i}, V_{2,i} > \quad \text{and } \bar{z_i} \neq \ \bar{z_j} \text{ for } 0 < i \neq \ j \leq \ 2$$

4.6.3 Security Considerations

Assuming, once again, that the underlying CS97 scheme is secure, we claim the following:

Lemma 4.6.1 *If $\mathcal{T}$ is a valid SGS by sub-group $S = \{M_1, ..., M_i\}$ on message m, then $\mathcal{T}$ has compositional integrity, i.e., anyone can verify that $\mathcal{T}$ is composed of i group signatures (of message m) each generated by a distinct group member.*

<u>Proof outline:</u> Since $\mathcal{T}$ is a valid SGS, for each constituent SIG_j, $\bar{z_j}$ is unique and all SKROOTLOG-s and SKLOGOLOG-s hold with a common base, $\bar{g}$. Suppose, by contradiction, that there exist two signatures $SIG_u, SIG_w \in \mathcal{T}$ that are actually generated by the same group (and also sub-group) member, M_k. Each group member has only one certificate of the form $(a^x + t)^d$.

For SIG_u, $V_{1,u}$ demonstrates knowledge of x_u where $\bar{z_u} = \bar{g}^{a^{x_u}}$ and $V_{2,u}$ demonstrates knowledge of the corresponding certificate $(a^{x_u} + t)^d$. However, SIG_w is also a valid group signature and $V_{1,w}$, $V_{1,w}$ demonstrate knowledge of x_w ($\bar{z_w} = \bar{g}^{a^{x_w}}$) and $(a^{x_w} + t)^d$, respectively. As M_k only has one certificate, $(a^{x_w} + t)^d = (a^{x_u} + t)^d$ and $x_w = x_u$. This means: $\bar{g}^{a^{x_w}} = \bar{z_w} = \bar{g}^{a^{x_u}} = \bar{z_u}$ which contradicts our assumption. □

Lemma 4.6.2 *It is computationally difficult to decide i) whether two different and valid SGS-s $\mathcal{T}_\infty, \mathcal{T}_\in$ are generated by the same sub-group or ii) whether sub-groups that produced $\mathcal{T}_\infty, \mathcal{T}_\in$ have any members in common.*

Proof sketch: Unlinkability of SGS-s, follows directly from the security of CS97. Although the same random base $\bar{g}$ is used in the generation of all single-member group signatures that form an SGS, each signer M_j uses a distinct membership key a^{x_j} in computing its $\bar{z}_j = \bar{g}^{a^{x_j}}$. The sequence:

$$\bar{g} = g^r,\ \bar{z}_1 = g^{r \cdot\ a^{x_1}}, \ldots, \bar{z}_i = g^{r \cdot\ a^{x_i}}$$

can be viewed as a sequence of ElGamal encryptions of a fixed message "1" where:

- (r, g^r) represent the one-time private and public-keys of the signing sub-group
- every $g^{a^{x_j}}$ $(0 < j \leq\ i)$ is the "public-key" of a given signer, M_j and a^{x_j} is the corresponding "private key".
- $\bar{z}_j = g^{r \cdot\ a^{x_j}}$ is the actual *encryption.*

Since r is unique and random for every SGS, unlinkability is achieved, however, only **outside the sub-group** as discussed in the next section.

4.6.4 Unlinkability Reconsidered

Having established SGS as a practical construct, we now observe that unlinkability, in the context of SGS, has acquired a new dimension:

> Unlinkability outside the signers' sub-group as opposed to universal unlinkability

The former refers to unlinkability of multiple signatures by **outsiders**, i.e., non-members of the signing sub-group. Whereas, the latter means that SGS-s are unlinkable by anyone with the sole exception of the group manager. To be more specific, consider the following two scenarios:

1. A sub-group $\mathcal{M}$ of like-minded group members (fully aware of each other's identities) decide to sign two documents. No one outside the sub-group (whether group member or not) can link the two SGS-s. However, since members of $\mathcal{M}$ "know" each other, they can implicitly link multiple SGS-s that they produce.
2. All members of a large group vote on two different issues. We consider only the YES votes. The YES-voters on the first issue, form a *de facto* sub-group $\mathcal{M}_1$ and produce SGS_1. Likewise, YES-voters on the second issue, form $\mathcal{M}_2$ and produce SGS_2. Since each member votes independently, the composition of $\mathcal{M}_1$ (and $\mathcal{M}_2$) must be secret

even to its own members. A member of $\mathcal{M}_1$ ($\mathcal{M}_2$) needs to know only that s/he signed and the total number of signers.

Furthermore, if we let $\mathcal{M}_3 = \mathcal{M}_1 \cap \mathcal{M}_2$, then, clearly it is evident that $0 \leq \#\mathcal{M}_3 \leq min(\#\mathcal{M}_1, \#\mathcal{M}_2)$. However, a group member $M_j \notin \mathcal{M}_1 \cup \mathcal{M}_2$ cannot narrow $\#\mathcal{M}_3$ any further. Also, a group member $M_j \in \mathcal{M}_1 \cup \mathcal{M}_2$ can at most narrow $\#\mathcal{M}_3$ to:

$$1 \leq \#\mathcal{M}_3 \leq min(\#\mathcal{M}_1, \#\mathcal{M}_2)$$

Put another way, **a signer cannot be linked across multiple SGS-s even by fellow co-signers.**

It is notable that the sample SGS construct of the previous section only provides unlikability of the first kind, i.e., multiple SGS-s can be linked but only by common co-signers. Recall that in the initial stage of our SGS scheme, signers agree on a common r (generated perhaps as a result of a Diffie-Hellman key exchange). Because r is known to all signers, given:

$$z_j = \bar{g}^{a^{x_j}} = (g^r)^{a^{x_j}}$$

produced by some signer M_j, any other signer M_i can compute: $(z_j)^{r^{-1}} = (g)^{a^{x_j}}$. The latter is constant for a given M_j and can thus be linked with other SGS-s where both M_i and M_j are co-signers.

The above may not be a problem in some practical settings since co-signers might choose to identify themselves to one another prior to signing (for other reasons). On the other hand, it turns out to be easy to change our example SGS scheme to provide universal SGS unlinkability.

We first note that there is no requirement in CS97 for a signer to know r used to compute the one-time random base $\bar{g} = g^r$. This allows us to amend the initial stage of SGS generation so that signers compute a common:

$$\bar{g} = g^r \text{ where } r = r_1 \cdots r_i \text{ and } r_j \in_R \mathbb{Z}_n^* \text{ by } M_j$$

The differences with respect to the first SGS scheme are superficial. The same group Difie-Hellman key agreement can be used, except that, instead of computing a common r, the signers compute a common g^r. (And, r itself is unknown.) However, in the resulting scheme, SGS-s are unlinkable even by common so-signers.

4.6.5 Synchronous *vs* Asynchronous Operation

Our sample CS97-based SGS scheme assumes synchronous or on-line operation. This is due to the nature of the initial stage: members of the signing sub-group (co-signers) agree on a secret r which is contributed to by each co-signer. Consequently, the scheme fails to address certain SGS scenarios, such as:

> Suppose, once again, that members of a large and distributed group are asked to vote. Instead of voting on-line, they are given a time window (say, a week) to cast their votes.

In this case it is clearly impractical to conduct the initial stage as described above. Ideally, each signer should be able to sign asynchronously, at its own leisure. At the same time, the crucial properties of SGS must be maintained; especially, compositional integrity.

One possible solution is to ask a trusted entity to generate a common random base $\bar{g} = g^r$ for all members who choose to sign a given message. This entity can be the group manager who is already trusted with issuing membership certificates and generating group public and group private keys. As noted in the previous section, signers do not need to know r; they must be assured that it is random and fresh. It seems reasonable to trust the group manager with this task.

More concretely, the group manager generates a one-time random base $\bar{g}$ and commits to it by signing (it) with its own private key which is unrelated to the group private key. Furthermore, the signed $\bar{g}$ must be associated with the specific message to be signed. In the voting example, the group manager would generate two signed statements (where SK_{gm} is the group manager's private key):

1. { YES, m=[text], $\bar{g_1}$, GROUP-ID } SK_{gm}
2. { NO, m=[text], $\bar{g_2}$, GROUP-ID } SK_{gm}

The advantage of this approach is the apparent lack of additional overhead imposed by SGS: the total cost is exactly that of computing multiple CS97 group signatures. The main drawback is, of course, the involvement of a trusted third party; whether the group manager or another trusted entity.

4.7 Summary

In conclusion, this chapter presented new concepts and authentication services unique to groups. In the second part of the chapter we presented two new and practical services that can be easily built on top of any group signature scheme: multi-group signatures (MGS) and sub-group signatures (SGS). After analyzing their respective security requirements, we presented sample constructs for both MGS and SGS.

Chapter 5

CLQ_API: A Group Key Agreement API

As collaborative applications grow in popularity the need for appropriate security guarantees, services and mechanisms becomes apparent. This chapter describes a protocol suite and an API geared for securing collaborative applications. The API is based on the extensions of Diffie-Hellman key agreement developed in the CLIQUES project. Its core services provide authenticated group key agreement in relatively small (on the order of 100 members) dynamic peer groups. The results described in this chapter have appeared in [10, 11].

5.1 Introduction

Securing group communication is a complex issue. It poses a number of challenges ranging from basic cryptographic algorithms to systems, communication design and secure implementation [39]. The two common group security concerns are: 1) privacy: assuring that intra-group communication remains secret to non-members, and 2) authentication – assuring that legitimate group members can be identified as such.

As we have seen in previous chapters, the standard approach to supporting group security is based on maintaining a secret quantity known only to all current group members The particulars of generating and distributing this secret quantity are known collectively as *group key establishment*. When the latter is achieved with one party generating a group secret, the problem is reduced to *group key distribution*. Whereas, if all group members collectively generate the group secret, the problem is referred to as *group key agreement*. In both cases, only current group members must have access to the group secret. We say *current* since group membership can be highly dynamic. Whenever members join or leave a group, there must be means for securely adjusting the group secret.

This chapter discusses the design of an Application Programming Interface (API) for group key agreement. This API, called CLQ_API , is based on the CLIQUES protocol suite and is geared specifically for dynamic peer groups. (Peer groups are relatively small, non-hierarchical groups typically used for replication or collaborative, many-to-many applications.) CLIQUES protocols [102, 103, 14], in turn, are the group extensions of the well-known Diffie-Hellman key exchange [49]. CLIQUES provides authenticated contributory key agreement which guarantees key independence, key confirmation, perfect forward secrecy, and resistance to known key attacks.

CLQ_API separates cryptographic protocols from communication, i.e., the actual group communication is left to the underlying communication subsystem (preferably, a reliable group communication system, e.g., SPREAD[6], TOTEM[81] or TRANSIS[3]). Moreover, network events such as network partitions, failures and other abnormalities are assumed to be taken care by the same communication system. As described in subsequent sections, this allows the design of a small, concise and communication-independent API.

The rest of this chapter is organized as follows. We begin with the notation and the brief description of basic operations in group key agreement. We then describe the CLIQUES protocol suite, which, in turn, consists of the following group operations: join, merge, leave and key refresh. Next, CLQ_API primitives are described in detail along with some examples. Finally, we discuss the efficiency of CLQ_API and conclude with some experimental results obtained with several popular cryptographic packages. (A more detailed description of the API is included in the last sections)

5.2 Group Key Agreement

The following notation is used:

5.2.1 Group key agreement operations

A comprehensive group key agreement API must handle adjustments to the group secret(s) stemming from single- and multiple-member membership population changes. This section describes the purpose of each of these operations.

5.2.1.0.1 Single-member changes: include single member additions or deletions. The former occurs when a prospective member wants to join a group and the latter –

n	number of protocol parties (group members)
i, j	indices of group members
M_i	i-th group member; $i \in [1, n]$
G	unique subgroup of Z_p^* of order q with p, q prime
q	order of the algebraic group
α	exponentiation base; generator in group G
x_i	long-term private key of M_i
N_i	M_i's session random number $\in Z_q$
S_n	group shared key among n members
K_{ij}	long-term shared secret between M_i and M_j, $i \neq j$
$\|a\|$	bit length of integer a
H	output size of hash function
$inv(a, b)$	multiplicative inverse of a modulo b

when a member wants to leave (or is forced to leave) a group. Both operations may be performed by the group controller(s) or by consent of all group members, depending on the local policy.

5.2.1.0.2 Multiple-member changes: also include addition and deletion. The former can be further broken into:

- Mass join: multiple disparate new members are brought into an existing group
- Group fusion: two or more groups are merged to form a single group.

The latter includes:

- Mass leave: multiple members must be removed at the same time.
- Group fission: a monolithic group needs to be broken into smaller groups.

5.2.1.0.3 Key refresh: is not a membership change operation, however, we discuss it here for the sake of completeness. It has two main purposes:

- Limit the amount of ciphertext generated with the same key.
  Since it is easier to perform cryptanalysis with more ciphertext/plaintext pairs, a routine group key refresh operation is needed. The lifetime of a key is determined by the application-specific policy.

- Recover from the compromise of a current group secret or a member's contribution. (We note that a compromise of a member's contribution can result in disclosure of all group secrets contributed to by this member. Therefore, not only the group shared keys, but also the individual key shares must be periodically refreshed.)

5.3 CLIQUES Protocols

As mentioned above, CLIQUES is a protocol suite providing authenticated contributory key agreement for dynamic peer groups. The following operations are supported by CLIQUES:

- Join: a new member is added to the group.
- Merge: one or more members are added to the group.
- Leave: one or more members are removed from (or leave) the group.
- Key Refresh: generates a new group shared key.

Each operation is discussed in the remainder of this section. The mapping between the group membership changes and the corresponding key agreement (cryptographic) operations is as follows:

Membership Operation	CLIQUES Operation
Forced leave	Leave
Voluntary leave	Leave
Single join	Join
Mass join	Merge
Group fusion	Merge
Group fission	Leave
Mass leave	Leave
Key refresh	Key Refresh

5.3.1 Notes

5.3.1.0.4 Group Controller. In the current version of CLIQUES, the last member to join the group becomes a *group controller*. Equivalently, the role of a group controller is always played by the newest member. We emphasize that a group controller is not in any way a privileged group member. It exists to prevent contention and should be viewed as a burden (or a chore) rather than a privilege. Specifically, the group controller assists new members in joining the group and initiates operations stemming from departing members or the need for periodic re-keying.

This behavior can be easily changed if required. In fact, any selection criteria can be used to select a group controller as long as it yields a consistent outcome across all group members. (CLIQUES assumes that the underlying communication system provides a timely and consistent membership view to all group members; this property is referred to as membership *view synchrony*.) One obvious alternative to the present criterion is to let the oldest member be the group controller.

Furthermore, it is important to note that *any* group member can, at any time, initiate a group re-key by unilaterally updating its key share and distributing appropriate partial keys to the rest for the group. (See below.)

5.3.1.0.5 Group Key. In the following operations, the group key (also referred to as group secret) has the form $S_n = \alpha^{N_1 \cdots\ N_i\ \cdots\ N_n}$, where n is the group size and N_i $(0 < i \leq\ n)$ is provided by the i-th member, M_i[102]. In all CLIQUES protocols, the last broadcast message is always the set formed by:

$$\alpha^{K_{in}\frac{N_1 \cdots\ N_n}{N_i}} \quad \forall i \in [1, n[$$

This set, from the last broadcast, must be retained by each group member. This is necessary since any member can become a controller due to a group partition or a network fault. (In other words, a non-controller can become a group controller if all "younger" members fail or become partitioned out.) Although the entire group receives this message from the current controller, each member uses a different element (key) to compute the new group secret.

Finally, the computation of the pairwise long-term key, K_{ij}, shared between any two members M_i and M_j is assumed to be performed before its use is required in the protocol.

5.3.2 Join

The join operation adds a new member, M_{n+1}, to the current group of n members. During this process a new group shared key, S_{n+1}, is computed and M_{n+1} becomes the new group controller. Assuming that M_n is the current controller, the protocol runs as follows:

1. M_n generates a new secret and random exponent N'_n and produces the following set:[1]

$$\mathcal{M} = \{\alpha^{N_1 \dots N'_n / N_i} \mid i \in [1, n-1]\}$$

[1]This is achieved by exponentiating $N'_n * inv(K_{in}, p) * inv(N_n, q)$ to each element in the last broadcast message.

Next, $\mathcal{M}$ is sent to M_{n+1}.

2. Upon receipt of the message, M_{n+1} generates a new secret exponent N_{n+1} and computes:
$$\mathcal{M}' = \{\alpha^{K_{i,n+1}N_1 \ldots N'_n N_{n+1}/N_i} \mid i \in [1, n]\}$$
This set is broadcast to the entire group (including itself).

3. Upon receipt of the broadcast, each M_i computes the shared group key as follows:
$$(\alpha^{\frac{N_1 \ldots N'_n N_{n+1}}{N_i} \cdot K_{i,n+1}})^{K^{-1}_{i,n+1} \cdot N_i} = \alpha^{N_1 \ldots N'_n N_{n+1}} = S_{n+1}$$

Steps 1 and 2 require n modular exponentiations (by M_n and M_{n+1}, respectively) and step 3 requires a single exponentiation by each member. Hence, the protocol requires $(2n+1)$ serial exponentiations.

5.3.3 Merge

The merge operation adds $k > 0$ members to the current group of $n > 1$ members. Let $m = n+k$. During this process a new group shared key, S_m, is computed and M_m becomes the new group controller. Assuming M_n is the current controller, the protocol runs as follows:

1. M_n generates a new exponent N'_n and computes: $\alpha^{N_1 \ldots N_{n-1} N'_n}$ by modular exponentiation of the previous group shared key with $(N'_n * inv(N_n, q))$. Then, this value is sent to M_{n+1}, the first new member.

2. Each new (merging) member M_j, $j = n+1, \ldots, m-1$, generates a new exponent N_j, computes $\alpha^{N_1 \ldots N'_n \ldots N_j}$ and forwards the result to M_{j+1}.

3. Upon receipt of the accumulated value, M_m simply broadcasts it to the entire group.

4. Every group member M_i (old and new) received the broadcast, computes $\alpha^{N_1 \ldots N'_n \ldots N_{m-1}/N_i}$ and sends back to M_m.

5. Having received all responses from the group members, M_m generates a new secret exponent N_m and produces the following set: [2]
$$\mathcal{M} = \{\alpha^{K_{im} N_1 \ldots N'_n \ldots N_m / N_i} \mid i \in [1, m-1]\}$$
which it broadcasts to the group.

[2]Note that $\mathcal{M}$ is the same as its counterpart in the Join protocol.

6. Each member M_i computes the group shared key exactly as in the last step of the Join protocol.

In case of a single-member Merge ($k = 1$), step 2 is not required and the rest of the protocol runs as above.

Steps 1 and 2 require a total of k modular exponentiations while steps 4 and 6 each involve one exponentiation (in parallel by each member). Finally, step 5 needs $(n + k - 1)$ exponentiations. Thus the total number of serial exponentiation for a k-member Merge is $(n + 2k + 1)$.

5.3.4 Leave

The leave operation removes one or more members from the group of n members. As a result, a new group shared key S_{n-k}, is computed where k is the number of the departing members. Let the set $\mathcal{K}$ denote the leaving members.

There are two cases to consider: 1) the current controller $M_n \in \mathcal{K}$ and 2) $M_n \notin \mathcal{K}$. In the first case, the controller's role is passed on to the most recent remaining member M_d where $d = n - k$. (See Section 5.4.1 below.) Assuming that members are indexed in the order of joining the group (i.e., M_n is the most recent member to join) the remaining members must be *renumbered* after a Leave. Now, even if the previous controller is not in $\mathcal{K}$, it will acquire a new index and become M_d due to renumbering.

The protocol runs as follows:

1. The controller M_d generates a new exponent N'_d, produces the following set and broadcasts it to the remaining group.

$$\mathcal{M} = \{\alpha^{K_{i,d} N_1 \ldots N'_d / N_i} \mid M_i \notin \mathcal{K}\}$$

2. Having received $\mathcal{M}$, each M_i computes:

$$S_d \; = \; (\alpha^{\frac{N_1 \ldots N'_d}{N_i} \cdot K_{i,d}})^{K_{i,d}^{-1} \cdot N_i} = \alpha^{N_1 \ldots N_d}$$

Members of the departing set $\mathcal{K}$ cannot compute the new key since the group controller only computes partial keys for the remaining group.

Step 1 requires $n - k$ exponentiations whereas step 2 needs a single exponentiation by each member. In total, the leave operation requires $(n - k + 1)$ serial exponentiations.

5.3.5 Key refresh

The key refresh operation updates the current group shared key, S_n. It is, in fact, a special case of a Leave with $k = 0$. Its usage depends on the application policy.

5.4 CLQ_API

CLQ_API is a group key management API based on the CLIQUES protocol suite. It is fairly small, containing only eight function calls.

CLQ_API does not perform any communication. All API calls either accept input tokens or produce output tokens (or both). Input tokens are assumes to have arrived as messages from other members while output tokens are messages to be sent to other members. As mentioned before, CLIQUES (and CLQ_API) requires reliably and sequenced communication and membership view synchrony. Any communication system providing these features can be used in conjunction with CLQ_API , e.g., SPREAD, TOTEM, and TRANSIS might be used.

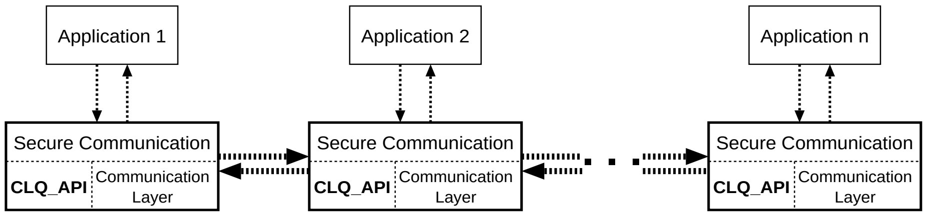


Figure 5.1: Communication layer and CLQ_API

5.4.1 CLQ_API Calls

This section describes CLQ_API calls. Each call represents one or more steps in one of the group operations already presented. Details of the data structures, definitions and secondary function calls can be found in the last sections of this chapter.

The following terms are used throughout this section:

- *Context:* contains information about each user required by CLQ_API.
- *Epoch:* message sequence number of a token. It is used to keep track of the group secret evolution and to prevent replay attacks.

- *Timestamp:* time when a token is sent. It is used to check the freshness of an incoming token.
- *Session random:* secret key share generated by each member.
- *Token:* basic protocol unit; includes: epoch, timestamp, and a CLIQUES protocol message.

We now describe the CLQ_API calls:

- clq_proc_join: performs *step 1* of the join operation. The controller calls this function to hand over information about the current group to a new member who will eventually become the new controller. The purposes of this function are:
  - generate a new session random for the current controller.
  - remove long term keys and previous session random from the partial keys of all users.
  - add the new session random into the partial keys of all users.
- clq_join: performs *step 2* of the join operation. The new group member calls this function using the token received from the current controller. The purposes of this function are:
  - generate a new session random for the new member.
  - generate long-term keys between the new member and each user (can also be done before calling this function).
  - compute the partial keys for other group members.
- clq_update_ctx: performs the *last step* of the join, merge, leave, and key refresh operation. Every member calls this function in order to update the group shared key upon receipt of the token sent by the current controller. The main purpose of this function is to compute group shared key with the caller's session random and the incoming token.
- clq_update_key: performs *step 1 and 2* of the merge operation. This function is called by the current controller and by all (but last) of the new members to add their session randoms to the group shared key.

  The main purposes of this function are:
  - generate new session random.
  - add new session random to the group shared key.

- clq_factor_out: performs *step 4* of the merge operation. Every group member (except the controller) calls this function to factor out its own session random number from the group shared key.
- clq_merge: performs *step 5* of the merge operation. The last new member (new controller) calls this function to add its session random to each of received partial keys from the group members.
- clq_leave: performs *step 1* of the leave operation. Every group member calls this function right immediately after one or more members leave the group. A group member may also become the current controller after calling this function; this can happen if it finds itself to be the newest member within the remaining group. The main purpose of this function is to remove the information of the leaving members. If the controller calls this function, then two more steps need to be performed:
  - generate new session random.
  - compute partial keys for other group members (except, of course, the departed ones).
- clq_refresh_key: performs *step 1* of the key refresh operation. The current controller calls this function, when the group shared key needs to be updated. The main purposes of this function are:
  - To generate new session random number.
  - To compute new partial keys by adding this session random number and removing old one.

5.4.2 Operations of CLQ_API

Before we explain each of the group operations in the API, we need to define some message types:

- NEW_MEMBER : sent by the controller to a new member, when a new member joins the group.
- KEY_UPDATE_MESSAGE : sent to every member in order to update the group shared key.
- MERGE_KEY_UPDATE : sent to every member to update the group shared key in merge operation.

- MERGE_BROADCAST : broadcast from the last new member when one or more members merge.
- MERGE_FACTOR_OUT : sent by each group member to the new controller in a merge operation.
- MASS_JOIN : sent to the next new member in a merge operation.

We now clarify group operations using the function calls from the previous section.

- Join

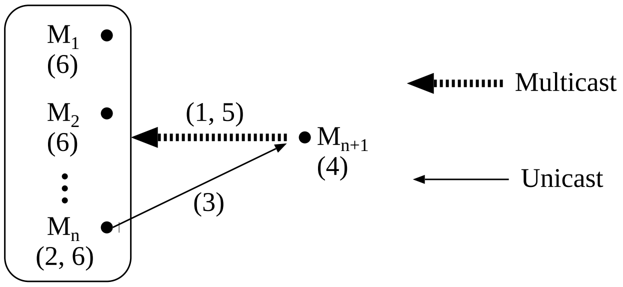


Figure 5.2: Join operation

(1) New member M_{n+1} calls communication protocol JOIN primitive.
(2) The current controller calls clq_proc_join to generate a token containing NEW_MEMBER message.
(3) The token is sent to the new member.
(4) The new member calls clq_join to generate a token containing KEY_UPDATE_MESSAGE.
(5) The token is broadcast to the entire group.
(6) Every user calls clq_update_ctx to compute the new group shared key.

- Merge (single member)

(1) New member M_{n+1} calls communication protocol JOIN primitive.
(2) The current controller calls clq_update_key to generate a token containing MASS_JOIN message.
(3) The token is sent to the new user (who becomes the new controller).

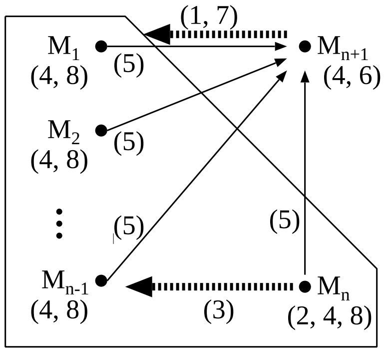


Figure 5.3: Merge operation

(4) The new controller calls clq_update_key to generate a token containing MERGE_BROADCAST message.

(5) The token is broadcast to the group.

(6) Each member (except the new one) calls clq_factor_out to generate a token containing MERGE_FACTOR_OUT message.

(7) Each member sends the output token back to the new controller.

(8) For each received token, the new controller calls clq_merge to generate a token containing MERGE_KEY_UPDATE message. When the last token is received, MERGE_KEY_UPDATE returns an output token.

(9) The token is broadcasted to the entire group.

(10) Every user calls clq_update_ctx to compute the new group shared key.

- Merge (multiple members)

(1) A MERGE event occurs either at an explicit request or as a result of a multiple members calling communication protocol join to join a group.

(2) The current group controller calls clq_update_key to generate a token containing MASS_JOIN message.

(3) The token is sent to the first merging member.

(4) Upon reception of the token, the next user calls clq_update_key to generate a token containing MASS_JOIN message.

(5) The token is sent to the next new member.

(6) Upon reception of the token, the last user calls clq_update_key to generate a token containing
MERGE_BROADCAST message.

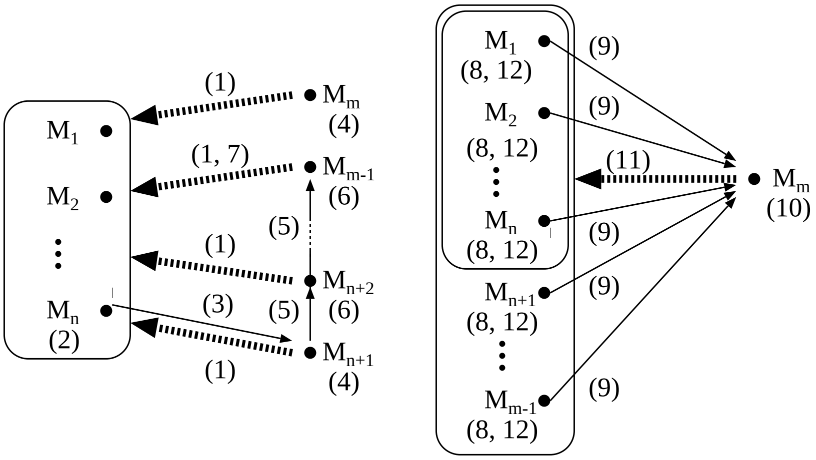


Figure 5.4: Merge(several members) operation

(7) The token is broadcasted to the entire group.

(8) Upon reception of the message, each member except the last one calls clq_factor_out to generate a token containing MERGE_FACTOR_OUT message.

(9) The token is sent back to the last new user.

(10) For each output token, the last new member calls clq_merge to generate a token containing MERGE_KEY_UPDATE.

(11) The token is broadcasted to the entire group.

(12) Every user calls clq_update_ctx to compute the new group shared key.

- Leave

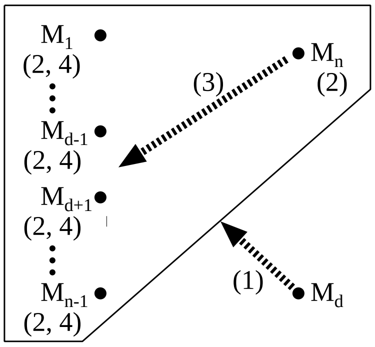


Figure 5.5: Leave operation

(1) The leaving member M_d calls communication protocol LEAVE primitive. (In case of a member disconnect or network failure the communication system notifies the group of the event.)

(2) All group members call clq_leave; only the group controller obtains the output token containing
KEY_UPDATE_MESSAGE.

(3) The token is broadcast to the entire group.

(4) Upon reception of the token, every user calls clq_update_ctx to compute the new group shared key.

- Key refresh

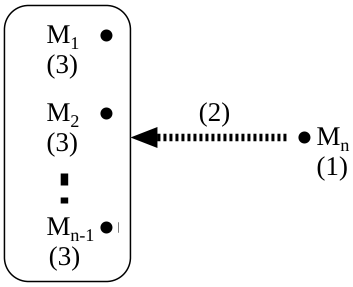


Figure 5.6: Key refresh operation

(1) The controller calls clq_refresh_key to generate a token containing
KEY_UPDATE_MESSAGE.

(2) The token is broadcasted to the entire group.

(3) Upon reception of the token, every user calls clq_update_ctx to compute the new group shared key.

5.5 Efficiency

The communication overhead is summarized in Table 5.1.

Table 5.2 illustrates computation costs. Exponentiation is the most expensive operation as it requires $O(log^3 p)$ bit operations in $\mathbb{Z}_p^*$. Given a and p, finding the inverse of $a \in \mathbb{Z}_p^*$ requires only $O(log^2 p)$ bit operations (using the extended Euclidean algorithm). Similarly, the multiplication of a and b modulo p requires $O(log^2 p)$ bit operations. See [68, 79] for more details. Hence, the speed of each operation depends largely on the number of serial exponentiations. (Note that the cost for generating the long term keys is not included in this table.)

Table 5.1: Communication costs

Operations	Join	Merge	Leave	Refresh	k-Merge	k-Leave
Number of users after operation	n+1	n+1	n-1	n	n+k	n-k
Rounds	2	3	1	1	k+2	1
Broadcasts	1	2	1	1	2	1
Reverse broadcast	0	1	0	0	1	0
Total messages	2	n+2	1	1	n+2k	1
Maximum bandwidth	n	n	n-1	n-1	n+k-1	n-k

Table 5.2: Computation costs

Operations	Join	Merge	Leave	Refresh	k-Merge	k-Leave
Number of users after operation	n+1	n+1	n-1	n	n+k	n-k
Serial exponentiation	2n+1	n+3	n-1	n	n+2k+1	n-k
Total exponentiation	3n+2	3n+2	2n-3	2n-1	3n+4k-2	2n-2k-1

5.5.1 Exponentiation

As mentioned earlier, modular exponentiation is the greatest overhead factor in CLIQUES protocols. In this section, we compare the performance of modular exponentiation operations with three different cryptographic libraries on three different processors:

Table 5.3: Target platform

Machine	CPU speed	Main memory	OS	Compiler
Ultra Sparc I	233 MHz	64 MB	SUN OS 5.5.1	gcc 2.7.2.2
Pentium I	233 MHz	48 MB	Linux 2.0.36	gcc 2.7.2.3
Pentium II	450 MHz	256 MB	Linux 2.2.9	egcs 1.1.2

The cryptographic libraries are summarized as follows:

- RSAREF[95]
  A cryptographic toolkit for privacy-enhanced mail. We used version 2.0 developed in 1996.
- Crypto++[47]
  A public domain C++ class library of cryptographic schemes published by Wei Dai. Note that, since addition and subtraction are implemented for the Pentium assembler,

this package performs better on Pentium than in other microprocessors. We used version 3.1 developed in May 1999.

- OpenSSL[86]
  A successor of SSLeay[62], OpenSSL is a cryptographic toolkit implementing Secure Socket Layer(SSL v.2/3)[60]. (Implements some basic operations in assembler on various platforms.) We used version 0.9.3a developed in May 1999.

We measured the performance of modular exponentiation $y = \alpha^x \pmod{p}$, where p is a random 512-bit prime, g a 512-bit generator for GF(p) of order q(160 bit), and x a random 160 bit integer. Table 5.4 shows the comparison.

Table 5.4: Comparison results

Machine	OpenSSL	Crypto++	RSAREF
UltraSparc	16.6 msec	60.2 msec	146.3 msec
Pentium II	2.5 msec	7.6 msec	22.2 msec
Pentium I	6.4 msec	N/A	N/A

At the time of this writing, OpenSSL appears to be the fastest of all public domain cryptographic libraries. On a Pentium II processor it requires only 2.5 *msecs* for each 512-bit modular exponentiation.

5.6 Data Structures and Definitions of CLQ_API

- CLQ_CONTEXT
  - Type : Structure
  - Description : This structure contains the context of a member, M_i, in a specific group.

- Contents

Name	Type	Contents
member_name	String	Name of the user M_i
group_name	String	Name of the group
key_share	$\|q\|$ bit integer	M_i's session random number, N_i
group_secret	$\|p\|$ bit integer	Current group shared key
group_secret_hash	H bit integer	Hash of group_secret
group_members_list	CLQ_GML	List of current group members
first	Pointer	Pointer to the first member in the group_members_list
last	Pointer	Pointer to the last member in the group_members_list
me	Pointer	Pointer to the member M_i in the group_members_list
params	CLQ_PARAM	Diffie-Hellman parameters
key	CLQ_KEY	Private and public key of M_n
epoch	Integer	Last message number used

- CLQ_GML
  - Type : Double linked list of CLQ_GM data structure
  - Description : This structure is a node of the group_member_list.
  - Contents

Name	Type	Contents
member	CLQ_GM	The current group member
prev	Pointer	Pointer to the previous node in the list
next	Pointer	Pointer to the next node in the list

- CLQ_GM
  - Type : Structure
  - Description : This structure contains information about a specific member.
  - Contents

Name	Type	Contents
member_name	String	Name of the member
long_term_key	$\|p\|$ bit integer	Long term shared key between myself and member_name, i.e. K_{ij} where i is related to the public key of member_name, and j to my private key
last_partial_key	$\|p\|$ bit integer	Last partial key for member_name.

- CLQ_TOKEN

– Type : Structure

– Description : Communication token used by CLQ_API, see also CLQ_TOKEN_INFO

– Contents

Name	Type	Contents
length	Integer	Size of t_data
t_data	Integer array	Contains the following encoded data: group_name, message_type, time_stamp, sender_name, epoch, group_members_list(without long_term_key)

- CLQ_TOKEN_INFO

  – Type : Structure

  – Description : This structure contains information about the token.

  – Contents

Name	Type	Contents
group_name	String	Name of the group
message_type	MSG_TYPE	Type of the message
time_stamp	Integer	Time stamp of the message
sender_name	String	Name of the sender

- CLQ_PARAM

  – Type : Structure

  – Description : Diffie-Hellman, DH, public parameters, i.e. p, q and g

  – Contents

Name	Type	Contents
p	$\|p\|$ bit integer	DH parameter p
q	$\|q\|$ bit integer	DH parameter q
g	$\|p\|$ bit integer	DH parameter g

- CLQ_KEY

  – Type : Structure

  – Description : Public key and private key of the user

  – Contents

Name	Type	Contents
priv_key	$\|q\|$ bit integer	Private key of the user
pub_key	$\|p\|$ bit integer	Public key of the user

5.7 API Calls of CLQ_API

- clq_join(ctx, member_name, group_name, input_token, output_token)
  - Caller : New member
  - Related to : Join
  - Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Context for the new user, this should be created in this function.
member_name	String	Name of the user calling this function
group_name	String	Name of the group that has been joined. This name has to match with the one that is included in the input_token.
input_token	CLQ_TOKEN	Message received from the controller. It is the output_token generated by clq_proc_join.
output_token	CLQ_TOKEN	New key update message to be broadcasted to the group. It will be used as input_token of clq_update_ctx.

- clq_proc_join(ctx, member_name, output_token)
  - Caller : Current controller
  - Related to : Join
  - Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current group context, ctx will be modified only if the caller is the controller.
member_name	String	Name of the new member
output_token	CLQ_TOKEN	This token should be used as the input_token for clq_join.

- clq_update_ctx(ctx, input_token)
  - Caller : Every group member.
  - Related to : Join, Merge, Leave, Key Refresh

- Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current context of each member
input_token	CLQ_TOKEN	Generated by new member or by the current controller, when an update of key is required(i.e. a user join, a user left, or the key has been compromised). It should be the output_token of clq_join, clq_merge, clq_leave, or clq_refresh.

- clq_update_key(ctx, member_list, input_token, output_token)
  - Caller : Current or new controller.
  - Related to : Merge
  - Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current group context
member_list	List of string	List of names of the new members. When a new member calls this function, this list should be null(Since the input_token is valid).
input_token	CLQ_TOKEN	Output of clq_update_key by the current controller or previous new member. When the controller calls this function, input_token should be null. (Since the member_list is valid).
output_token	CLQ_TOKEN	Contains refreshed group_secret. It will be used as input_token of * clq_factor_out by the current group members or * clq_update_key by the next new member.

- clq_factor_out(ctx, input_token, output_token)
  - Caller : Every group member (except the last one[3])

[3] If the current controller, i.e. the last member, calls clq_factor_out then the output_token will return NULL

- Related to : Merge
- Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current member context
input_token	CLQ_TOKEN	output_token of clq_update_key
output_token	CLQ_TOKEN	Contains updated last_partial_key by removing each user's key_share. It will be used as input_token of clq_merge.

- clq_merge(ctx, sender_name, input_token, output_token)
  - Caller : The last new member
  - Related to : Merge, Mass Join
  - Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current group context. ctx will be modified.
sender_name	String	Name of the sender of the input_token
input_token	CLQ_TOKEN	output_token of clq_factor_out
output_token	CLQ_TOKEN	Contains updated last_partial_key

- clq_leave(ctx, member_list, output_token)
  - Caller : Every group member
  - Related to : Leave
  - Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current group context
member_list	List of string	List of members leaving
output_token	CLQ_TOKEN	Updated message to be broadcasted to the group

- clq_refresh_key(ctx, output_token)
  - Caller : Controller
  - Related to : Key Refresh
  - Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current group context. ctx will be modified.
output_token	CLQ_TOKEN	Updated message to be broadcasted to the group

- clq_destroy_ctx(ctx)
  - Description : Frees the space occupied by the current context
  - Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current group context. ctx will be destroyed.

- clq_destroy_token(token)
  - Description : Frees the space occupied by the token
  - Parameters

Name	Type	Description
token	CLQ_TOKEN	Input or output_token

- clq_first_user(ctx, member_name, group_name)
  - Description : clq_first_user is called by the first user who joins a group
  - Main purposes
    * Generates key_share.
    * Generates member context.
  - Caller : The first member in a group
  - Related to : Join
  - Parameters

Name	Type	Description
ctx	CLQ_CONTEXT	Current member context. ctx should be created.
member_name	String	Name of the first user
group_name	String	Name of the group

5.8 Summary

This chapter described a protocol suite and an API designed specifically for securing dynamic collaborative applications in unreliable networks. The API is based on the extensions of Diffie-Hellman key agreement developed in the CLIQUES project. It provides core security services (most notably, authenticated key agreement) for relatively small and dynamic peer groups.

Chapter 6

Security in Asynchronous Networks with Failures

Increasing popularity and diversity of collaborative applications prompts the need for highly secure and reliable communication platforms for dynamic peer groups. Security mechanisms for such groups tend to be both expensive and complex and their integration with reliable group communication services presents a formidable challenge. This chapter reports on our implementation experience as appeared in [2]. I wish to thank in particular Cristina Nita-Rotaru, Theo Schlossnagle, John Schultz, and Jonathan Stanton for letting me report on their work and for providing me the necessary tables, timings, efficiency analysises, texts and ideas to substantiate my research statements.

6.1 Introduction

Fault-tolerant, scalable, and reliable communication services have become critical in modern computing. A major focus today is in taking traditional, centralized services such as file sharing, authentication, web, and mail services and distributing them across multiple systems and networks. These distributed applications and other inherently collaborative applications (such as conferencing, whiteboards, shared instrument control, as well as command-and-control systems) are very difficult to implement. One common and successful approach to developing these types of applications is to use a reliable group communication toolkit as a base messaging and fault-tolerant service.

Reliable group communication systems offer a set of low-level services that provide efficient messaging, membership, ordering and fault-detection services to peer groups. Typically, distributed and collaborative applications require fairly low latency message delivery of both small and large messages. They may involve many-to-many communication patterns. The number of membership events that these applications generate can vary from one every few minutes or hours to tens or hundreds of membership changes per second. Commonly, however, the number of joins or leaves is at most a few per second and network failures or recoveries causing merges of groups or partitions are at most a few an hour. Thus, we aim for the secure group communication system to support that level of performance in a practical setting.

The rest of this chapter is organized as follows. The next section outlines, in general terms, our security focus. Then, Section 6.4 provides an overview of the SPREAD group communication toolkit. Next, the secure SPREAD architecture is described in Section 6.5. Section 6.6 illustrates and discusses some experimental results obtained with secure SPREAD and Section 6.7 summarizes the related work. Section 6.8 concludes the chapter with the discussion of on-going and future work.

6.2 Our Security Focus

The need for specialized group security mechanisms has been already motivated by the previous chapters. Since routine security services such as bulk data privacy and data integrity are usually contingent upon sharing a common secret (group key), establishing and managing group keying material is the most fundamental group security mechanism.

We thus concentrate on group key management, its integration with a reliable group communication platform and its impact on the latter. We also consider the impact of data privacy and data integrity, however, these services are not much different in peer groups from those in a traditional two-party communication setting.

There are tradeoffs to be made when choosing what type of key management protocols a security system should use. These include number of messages sent per event, number of participants per event, amount of serial computation and overall computation done by the group per event, fault tolerance, amount of trust in members of the group and fairness of load distribution. We discuss these tradeoffs as we evaluate two group key management protocols: a distributed key management protocol suite based on the CLIQUES work, and a centralized protocol providing a roughly equivalent level of security.

In tackling the security issues in reliable group communication we do not (yet) consider certain components that are needed for comprehensive security.

- Group access control and, more generally, group policy
- Group and member certification

While we do not aim to underestimate their importance, the research in this area is just beginning [59]. We believe that well-designed group key management and data security protocols can be coupled with an *ad hoc* policy framework and be deployed rapidly. We further believe that the same group key management system can be coupled with a better policy framework when such exists.

6.3 Centralized vs Distributed Key Management

The main issues in group key management center around **who** generates a group key as well as **how** and **when** it is generated. The **when** part is relatively simple, since, in the extreme, a new key must be generated following every group membership change. The **who** and **how** issues are more involved as they collectively determine the actual key management protocol(s).

If the group key is generated by a single party who then distributes it to all other group members we call the key management *centralized*; whereas, if all members participate in key generation we call the key management *distributed*.[1]

Centralized key management takes on two flavors: TTP-based or controller-based. The first is based on the notion of a Trusted Third Party (TTP) which is a fixed, highly secure entity (e.g., a Kerberos Authentication Server [101]) charged with user authentication as well as generation and distribution of keys. Since a TTP is fixed, this approach cannot tolerate TTP partitions or TTP failures and is thus of limited utility in a peer group. An alternative is to fix a certain group member (e.g., oldest or newest) as a group controller whose duty is to generate and distribute keys to the group. Failures or partitions of a group controller can be effectively dealt with by selecting an appropriate member within the remaining group. This approach is workable and we are in fact using it as a point of comparison in the experimental results (see Section 6.6 and Appendix.) However, it does not allow for the authentication of individual group members which is one of our security goals. Furthermore, a fixed controller presents an attractive attack target since it is the only member entrusted with the security of the entire group. Another drawback is the inability to authenticate certain membership changes.

Distributed key management involves all group members collectively generating (or agreeing upon) a group key. The CLIQUES protocol suite falls into this category. A group secret is essentially a function of all group members' individual contributions. At the same time, a member's contribution is known only to that member; this property aids in the authentication of individual members (since a member can show that it knows a unique and

[1]For the sake of clarity we do not consider hybrid methods, i.e., a subset of members generating a group key.

secret share of a common group secret.) The often-cited drawbacks of distributed key management are the relative complexity and the computational overhead of the cryptographic protocols that implement it. On the other hand, as illustrated by experimental results in Section 6.6, the overhead is actually comparable to that of centralized controller-based key management.

6.4 The SPREAD Group Communication Toolkit

The group security services discussed in this chapter are built on top of the SPREAD wide-area group communication system [6].

SPREAD is a group communication system for local area and wide area networks. SPREAD provides all the services of traditional group communication systems, including unreliable and reliable delivery, FIFO, causal, and total ordering, and membership services with strong semantics.

SPREAD creates an overlay network that can impose any arbitrary network configuration including for example, point-to-multi-point, trees, rings, trees-with-subgroups and any combinations of them to adapt the system to different networking environments. The SPREAD architecture allows multiple protocols to be used on links between sites and within a site.

SPREAD is very useful for applications that need the traditional group communication services such as causal and total ordering, and membership and delivery guarantees, but also need to run over wide area networks.

In addition, other applications may find SPREAD useful because of some other technical properties:

- Scalability with the number of collaboration sessions. SPREAD can support large number of different collaboration sessions, each of which spans the Internet but has only a small number of participants. The reason is that SPREAD utilizes *unicast* messages on the wide area network, routing them between SPREAD nodes on the overlay network.
- Scalability with the number of groups. SPREAD can scale well with the number of groups used by the application without imposing any overhead on network routers. Group naming and addressing is no longer a shared resource (the IP address for multicast) but rather a large space of strings which is unique per collaboration session.
- SPREAD supports the Extended Virtual Synchrony model [80] and the View Synchrony model [50] which provide strong semantic guarantees about membership

events and message delivery.

- SPREAD uses a daemon-client architecture. This architecture has many benefits, the most important for wide-area settings is the resultant ability to pay the minimum necessary price for different causes of group membership changes. Simple join and leave of processes translates into a single message. A daemon disconnection or connection does not pay the heavy cost involved in changing wide area routes. Only network partitions between different local area components of the network requires the heavy cost of full-fledged membership change. Luckily, there is a strong inverse relationship between the frequency of these events and their cost in a practical system. The process and daemon membership correspond to the more common model of "Lightweight Groups" and "Heavyweight Groups"[RG98].

The SPREAD toolkit is available publicly. An early version of the system is used by several organizations for both research and practical projects. The toolkit supports cross-platform applications and has been ported to several Unix platforms as well as Windows and Java environments.[2]

6.4.1 Supported Group Communication Semantics

SPREAD supports the Extended Virtual Synchrony (EVS) model [80] [1] and the View Synchrony (VS) model [50]. Both EVS and VS guarantee that group members see the same set of messages between two sequential group membership events. They also both guarantee that the order of messages requested by the application (such as FIFO, Causal, or Total) is preserved. EVS provides a more general service, for VS semantics can be implemented on top of EVS semantics, but EVS can not be implemented on top of VS.

EVS provides two general benefits over VS: it has better performance and it allows open groups, where non-members of a group can send messages to the group. EVS guarantees that messages are delivered to all recipients in the same membership as the message was originally sent on the network.

The last EVS property above is the critical difference between VS and EVS. VS, in contrast, guarantees the stricter property that messages are delivered to all recipients in the same membership as the sending application thought it was a member of at the time it sent the message. Providing this property requires a round of application acknowledgement messages before installing a new membership. This need for application level acknowledgements requires that the groups be closed, only allowing members of the group to send messages to it.

[2]More details on the SPREAD system can be found at *http://www.spread.org/* along with a white paper and programming documentation.

This knowledge that a message is received in the membership the application believed it was sent in which is provided by VS semantics makes implementing a secure group system much easier because every message is encrypted with the same key as the receiver believes is current when the message is delivered to them.

The SPREAD system provides Extended Virtual Synchrony. A flush layer built atop SPREAD (provided with the SPREAD system) provides the View Synchrony model.

6.5 Secure SPREAD Architecture

As mentioned previously, SPREAD supports both VS and EVS group semantics. Currently secure SPREAD uses the VS group semantics. Also, SPREAD uses a daemon-client communication architecture. The current implementation of secure SPREAD is as a layer that extends the SPREAD client library and adds the new security features discussed in this chapter. Client applications then link with this secure library and use the provided API which is similar to SPREAD's API. A different way secure SPREAD could be implemented would be to graft the security features directly onto the SPREAD daemons themselves. These two approaches will be referred to as the client model and the daemon model, respectively. The benefits of each model are the drawbacks of the other.

The main advantage of the client model is that no trust is put into anything outside of the end user's control. The secure SPREAD client implements all of the security operations in the system. Because of this and the fact that the client is open source, an end user can easily verify for himself that the secure SPREAD client is trustworthy. Another advantage is that this particular design demonstrates how to secure any group communication system that supports the VS model.

The daemon model denies an end user access to the security implementation, especially if the daemon network is out of the end user's control, however, it does offer several other advantages. The main benefit gained from this approach is a large increase in performance. Keys would no longer need to be established for every group. Instead, the daemons could encode all of their intercommunication using one private key. Daemons are long lived entities and might only update their private key if their interconnectivity changed or a daemon crashed or recovered (daemons could also choose to rekey on a time out as well). These kinds of events occur rarely in practice and the number of daemons in a configuration is usually small. Therefore, in the daemon model the number of rekeyings occurring in the system as a whole would be drastically reduced when compared to the client model. Note, however, that communication between clients and daemons would still have to be secured in some manner. For example, a client could assume that IPC is secure, or instead use encrypted communications such as SSL sockets.

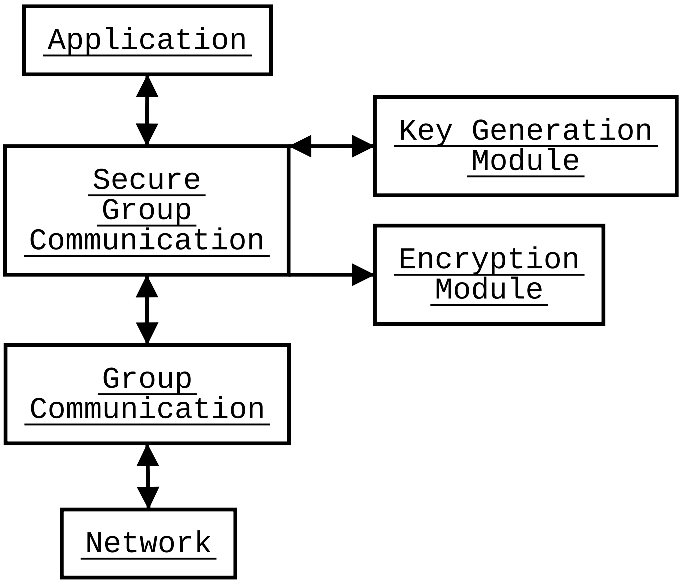


Figure 6.1: A Generic Secure Architecture

Another advantage of the daemon model is that security policy can be easily configured and enforced. In this model an administrator could change policy by editing a daemon configuration file. End users connecting to daemons could be authenticated upon connection. Access control to the daemons or specific groups could be easily enforced. Daemons could also easily authenticate that users are members of specific groups.

Despite all of the apparent advantages in using the daemon model, the client model has several security advantages over the daemon model. As stated above, no trust is placed in another entity, and furthermore, the system as a whole is less vulnerable to attack. For example, in the daemon model, if a daemon key is obtained, all messages in every group are compromised until the daemons rekey. In the client model, however, if a group key is obtained then only the messages in that group are compromised until the group rekeys. Due to the different levels of security offered by the two models there will be places for both approaches to be applied.

6.5.1 High Level Design

A major consideration when designing a secure communications toolkit is the security and trust that go along with the algorithms used in the toolkit. As time goes on, trust in these algorithms may change: ciphers are broken, algorithms are proven insecure, better algorithms are designed, etc. Therefore, any system hoping to secure communications and be viable in the long run, needs to be flexible and easily modified. Another desirable feature is if security policy is easily changed by administrators and/or users. One way to achieve these goals is to design an extremely modular system.

Figure 6.1 shows a basic architecture for a modular secure system. When necessary, the secure group layer calls different modules that implement encryption, key management, and key generation routines. If the secure layer only needs to know when to call these modules, but not their functions, then this type of design allows for easy drop-in replacement of different modules. Therefore, if a new module needed to be added to this type of layer, then only two modifications would have to be made. First, code must be written implementing the new module, and second this module must be chosen to run at the proper times.

6.5.2 Secure SPREAD Modules

The basic design of secure SPREAD follows this modular design closely. The core functionality of secure SPREAD is an event handling loop. Network events are generated by the VS group communication layer. Secure SPREAD takes each of these events and depending on their context, passes it on to the proper module that handles the event.

An interesting addition to the design in secure SPREAD is that the modules that implement the security policies and algorithms of the system can be chosen at run time as new groups are created. This mechanism allows different groups to choose different event handlers while simultaneously running in the same group communication system. For example, one group could decide to use centralized key management, while another group is using distributed key management at the same time.

Currently, secure SPREAD is designed to allow for easy drop in replacement of encryption and of key agreement protocols. In the future we hope to extend the architecture to easily allow for modules that would make access control and other policy decisions as well. At this early stage of development, secure SPREAD has only a few different modules that have been implemented.

Figure 6.2 shows the current actual implementation of secure SPREAD. The Flush layer provides View Synchrony semantics to the secure SPREAD layer. Note that the application can have several connections open at the same time and each of them can be in multiple groups. Some of these groups can be secure, while others can instead use the View Synchrony semantics of the Flush layer or the EVS semantics provided by SPREAD. Choosing which services are used is entirely up to the application.

Currently, for bulk data encryption secure SPREAD uses an open source implementation of Bruce Schneier's Blowfish algorithm [96]. In the near future we hope to add the ability to use the OpenSSL[86] cryptographic library which provides an abundant selection of encryption algorithms. With this selection of encryption algorithms available, the user

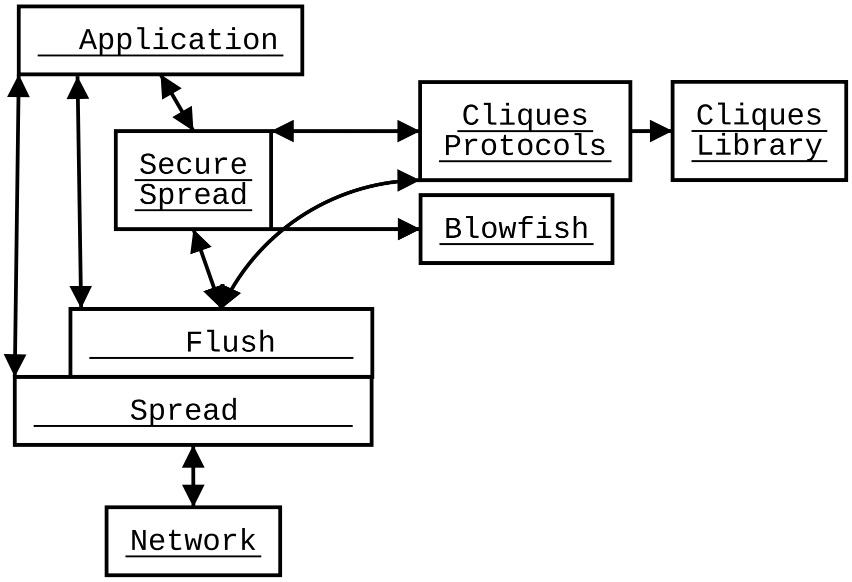


Figure 6.2: Secure SPREAD Implementation

could be allowed to decide which encryption algorithm they are most comfortable using.

Secure SPREAD currently has two different modules for key agreement, both based on the CLIQUES protocol suite.

- CLIQUES key management (using group Diffie-Hellman)
- Simple centralized key management (described in the Appendix)

The complexity of secure SPREAD is contained almost solely within these modules.

6.5.3 Implementation of CLIQUES Group Key Management

The CLIQUES key management suite and the corresponding API presented in chapter 5 are independent from a particular implementation of a group communication system.

How these protocols are implemented specifically in secure SPREAD is the subject of this section.

In order to easily accommodate the CLIQUES protocols, the underlying group communication system needs to provide certain fundamental features. These include: group multicast, group member to group member unicast, FIFO ordering on messages, and a mechanism for knowing and identifying all of the members of a group. All of these services are provided by SPREAD.

Furthermore, the CLIQUES API is tuned to allow for the following types of membership changes: singleton join, singleton leave, multi-join, and multi-leave. Again, all of these types of membership events are provided by SPREAD. In addition, the VS model of SPREAD generates an event that is a combination multi-join and multi-leave, and an event requesting the application to OK a group membership change.

To actually implement the CLIQUES protocols to be used with a group communication system a mapping must be made between group communication events and CLIQUES events. Presented in Table 6.1 is the simple mapping of SPREAD's VS group events to CLIQUES events.

Table 6.1: Mapping of SPREAD Events to Group Key Events

SPREAD VS Group Membership Events	Group Key Management Operations
Join	Join
Leave	Leave
Disconnect	Leave
Partition	Leave
Merge	Merge
Partition + Merge	Leave then Merge
Group Change Request	N/A
N/A	Key Refresh

When a VS group event is generated by SPREAD it is passed up to the security layer. Secure SPREAD then examines the event and determines to which group the message corresponds. It retrieves the event handler for that particular group and gives the handler the event. This handler is responsible for mapping the network VS event into the proper actions to be taken with respect to the current state of the group and the protocols it implements.

In our implementation, the CLIQUES protocols are implemented using state machines. The current state of a group member is associated with each process connection and a specific group. This state machine and the current state of the entity in the group will be collectively referred to as the "member state."

The current member state and the current event affecting the group determine what actions are taken by each member. The CLIQUES library keeps most of state information associated with a group, such as who is the current group controller. When calls are made to the CLQ_API, in general, all group members make the same function call and the return values of these calls determine how the member state of each member changes and what actions they take.

FIFO ordered messages were used for all messages used to communicate partial keys or key shares to the group or between particular entities. This was done because FIFO ordered messages have extremely low overhead associated with them, and stronger message orderings are not required.

6.5.4 Cascading Failure Handling

When merging security protocols with high reliability group communication protocols we must begin with an obvious premise that security does not imply robustness, i.e., a system can be secure but not robust. A security protocol does not become more secure when built over a reliable communication platform; it becomes more robust. For instance, many security protocols for group communication assume that messages are received in certain order or that all protocol parties are (at the same time) connected and ready to receive messages. In real systems these assumptions are often not fulfilled. Adding reliability through a group communication system can make these assumptions realistic and protocols themselves – usable.

Focusing on the problem at hand recall that – as discussed above – fluctuations in group channel state must be coupled with group key adjustments. However, adjustments require the availability and reachability of each group member. If the group channel state changes while an adjustment (stemming from an earlier change) is in progress, appropriate actions must be taken not to maintain security but to preserve the overall group integrity. This is in stark contrast to two-party communication where such "cascading" fluctuations simply do not occur.

Thus the greatest challenge in integrating reliable group communication and group security protocols lies not in straightforward mapping of well-spaced communication events into their security counterparts but in the proper and robust handling of various **cascading** events that perturb group channel state.

Thus far we have only implemented key agreement for non-cascading membership events. This allows us to benchmark the system and learn about its performance in essentially all of the common cases. However we do have solutions handling these cases that will be implemented in the near future. We discuss the ideas behind these solutions here.

The basic difficulty arises after the secure layer has received a new membership event from the VS layer and started to do whatever security work needs to be completed before the new membership configuration can be considered secure and the application can continue working. The security layer now has to send more messages and do computation to, for example, generate a new key if someone joined the group. However, during this time new membership events can occur, which might invalidate the current membership (say a partition occurred) or might necessitate more work (another member joined and also needs to be added to the secure group). Actually, an essentially unlimited number of such events can happen before the security layer has completed handling the first membership event. This implies that there is an upper limit on how many membership events can occur per unit time if the system is to perform any useful work.

All membership events can be divided into two types: those that affect the current secure membership handling, and those that can be safely deferred until the current event is handled.

For example, if someone joins a group and during the handling of the join someone else leaves, the leave must be included in the handling of the join or else the leaving member will either learn the key with the new member in it (which they should not know), or they have left because they died and if they are needed to complete the secure join then the system can deadlock (i.e. if they were the previous controller).

An example of a membership event type which is safe to defer, is a join to a group while a join of another member is being completed. Since the second join has no effect on a group that they have not officially/securely joined, the first join will be able to securely and successfully complete.

Note that one can not just use the VS properties which allow each application to delay the new membership until it is finished, because at the time the security layer is asked to OK a new membership change it does not yet know what the membership event is (Join/Leave/Partition) or who is involved. Thus, it can not know whether the event is safe to defer or not.

The security layer must OK the new membership. After OK'ing it and receiving the new membership the security layer evaluates the event and determines if it is deferrable. If it is, it queues it and continues with the security protocol. If it is not, then it follows a state dependent protocol that includes the new information into the current proposed secure membership and restarts the key agreement protocol based on the new secure group.

6.6 Experimental Results

In this section we present the experimental results obtained with Secure SPREAD. The experiments were performed in two different architecture sets under the same group scenario. In the first set, all machines were SUN Ultra-s 2 Model 1200 (200 MHz UltraSPARC, 128MB) running Solaris 5.5.1, while in the other set, all machines were Pentium II (450 MHz, 128MB) running RedHat Linux 2.2.7. The timings were obtained performing multiple batches of each operation 50 times and then averaging across batches. The setup consists of three identical machines, each running its own SPREAD daemon. Two machines had a single member each and the third machine contained all other members (processes) utilizing a single daemon. Machines were connected with an Ethernet 10 BaseT network for the SUN Ultra-2s and an Ethernet 100 BaseT network for the Pentium II machines. CLQ_API was linked with OpenSSL 0.9.3a [86], where one Diffie-Hellman (DH) exponentiation with 512-bit modulus costs 12 and 2.5 msecs for the SUN and Pentium platforms, respectively.

The total number[3] of serial exponentiations required for Join and Leave operations is illustrated in Tables 6.2 and 6.3, respectively. Table 6.4 summarize the total number of serial exponentiations performed in the course of each operation.

Table 6.2: Detailed number of exponentiation for Join

<table>
<tr><td rowspan="8">CLIQUES</td><td rowspan="4">Controller</td><td>Update key share with every member</td><td>$n-1$</td></tr>
<tr><td>Long term key computation with new member</td><td>1</td></tr>
<tr><td>New session key computation</td><td>1</td></tr>
<tr><td>Total:</td><td>$n+1$</td></tr>
<tr><td rowspan="4">New Member</td><td>Long term key computations</td><td>$n-1$</td></tr>
<tr><td>Encryption of session key</td><td>$n-1$</td></tr>
<tr><td>New session key computation</td><td>1</td></tr>
<tr><td>Total:</td><td>$2n-1$</td></tr>
<tr><td rowspan="10">CKD</td><td rowspan="5">Controller</td><td>Long term key computation with new member</td><td>1</td></tr>
<tr><td>Pairwise key computation with new member</td><td>1</td></tr>
<tr><td>New session key computation</td><td>1</td></tr>
<tr><td>Encryption of session key</td><td>$n-1$</td></tr>
<tr><td>Total:</td><td>$n+2$</td></tr>
<tr><td rowspan="5">New Member</td><td>Long term key computation with controller</td><td>1</td></tr>
<tr><td>Pairwise key computation with controller</td><td>1</td></tr>
<tr><td>Encryption of pairwise secret for controller</td><td>1</td></tr>
<tr><td>Decryption of session key</td><td>1</td></tr>
<tr><td>Total:</td><td>4</td></tr>
</table>

Figure 6.3 shows the total time of one join or leave operation, which costs the same, versus the group size.[4] The time in this graph includes the network overhead. The Flush layer timings in the graph are not linear due to the fact that n messages have to be broadcasted from each member to every other. This interaction caused a significant amount of overhead[5] in the machine that has most of the group members.

[3] During the join operation n includes the new member and during the leave operation n includes the leaving member.

[4] The Flush layer values on the Sparc architecture are not included.

[5] This overhead could have been caused by the operating system scheduler as well.

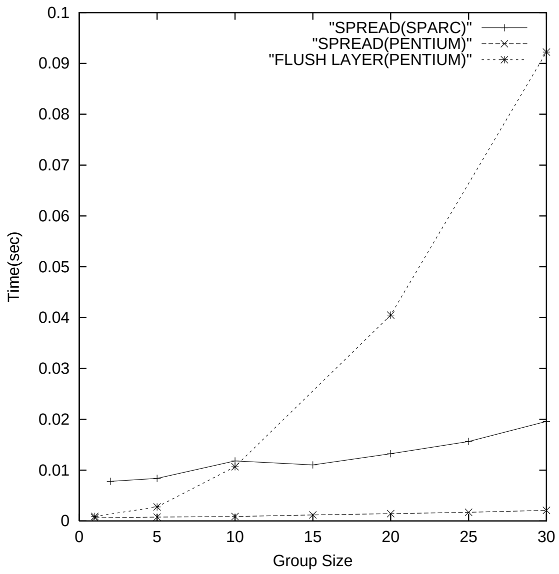
Table 6.3: Detailed number of exponentiation for Leave

CLIQUES	Remove long term key with previous controller	1
	New session key computation	1
	Encryption of session key	$n-2$
	Total:	n
CKD	New session key computation	1
	Encryption of session key	$n-2$
	Total:	$n-1$
CKD, when controller leaves	Long term key computations	$n-2$
	Pairwise key computation with new user	$n-2$
	New session key computation	1
	Encryption of session key	$n-2$
	Total:	$3n-5$

Table 6.4: Total number of serial exponentiation

Operation	Join	Leave	Controller leaves
Number of members after operation	n	$n-1$	$n-1$
CLIQUES	$3n$	n	n
CKD	$n+6$	$n-1$	$3n-5$

Figure 6.3: SPREAD and Flush Layer Timings



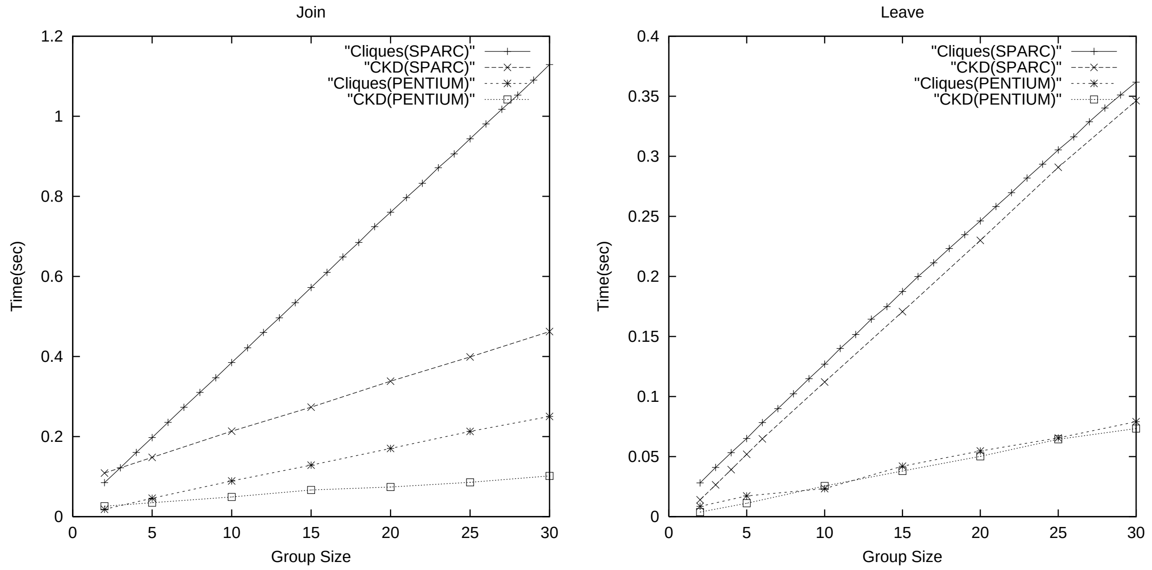


Figure 6.4: Timings

Figure 6.4 shows the total time of Join and Leave operations versus the group size in two separate graphs. The graphs seem to follow closely the total number of expected exponentiations in Tables 6.2 and 6.3. In other words, the results clearly show that exponentiation is the most dominant operation, as compared with network overhead, other arithmetic operations, and disk I/O. For example, a join operation in a group of fifteen members takes 0.1125 secs for the modular exponentiation and the total CPU time experimentally takes 0.1285 secs for the Pentium. Hence, 88% of the CPU was used for modular exponentiation.

6.7 Related Work

Related work falls into three categories: 1) cryptographic protocols for group key management, 2) architectures and frameworks for secure multicast and 3) implementation of secure group communication systems. Given the "systems" orientation of this chapter, we concentrate on (3). Readers interested in (1) are referred to [102] and [14]. We choose not to dwell on the related work in secure multicast since both its security and its communication models differ greatly from those of dynamic peer groups. Suffice it to say that it typically assumes one-to-many communication paradigm and emphasizes scalability over strong security; more specifically, the focus is on minimizing bandwidth overhead from re-keying a very large group.[6] Generation and distribution of group keys is assumed to be

[6]Most recent work in secure multicast has taken place under the aegis of the Internet Research Task Force's (IRTF) Secure Multicast Group (SMUG); see *http://www.ipmulticast.com* for further information.

performed by trusted servers which, as discussed above, is an approach unsuitable for a peer group setting.

Group communication systems in LAN environments have a well-developed history beginning with ISIS [23], and more recent systems such as Transis [4], Horus [93], Totem [5], and RMP [108]. These systems explored several different models of Group Communication such as Virtual Synchrony [22] and Extended Virtual Synchrony [80]. More recent work in this area focuses on scaling group membership to Wide Area Networks (WANs) [8].

Research in securing group communication systems is fairly new and is only beginning to be explored. The only actual implementations of group communication systems which focused on security issues are the secure distributed CORBA (Immune) system built on top of SecureRing[65] group communication work at UCSB [82] and Horus/Ensemble work at Cornell [94].

The Immune system from UCSB provides protection against Byzantine failures using cryptographic techniques to secure a low-level ring protocol (that forms the base of the Totem system) and replicating the protected CORBA objects sufficiently to detect and recover from upto a fixed number of compromised objects or machines.

The Ensemble security work exemplifies the state-of-the-art in reliable group communication security and addresses several of the same problems we consider in this chapter. It also allows application-dependent trust models and optimizes certain aspects of group key generation and distribution protocols. The problems we both address are in generating shared group keys and rekeying, however, our respective approaches are quite different. The Ensemble security protocols are tied into the Horus/Ensemble layered protocol stack architecture and interact with the group communication system at several levels, not just as an application. Ensemble also uses a different group key structure which is not contributory and provides a different set of security guarantees.

More specifically, Ensemble relies on *ad hoc* extensions of conventional (non group-oriented) security tools, namely PGP [109], to distribute and refresh group keys. The main drawbacks of such approaches are (recall that these are also summarized in Section 6.3):

- Generation of group keys is centralized (done by group leader) which goes contrary to the peer group model. In particular, the entire group relies on one member's ability to generate strong, secure keys. It is unclear how a group leader is selected/elected in the event of partition or failure.

- Key distribution is performed with the help of encryption (via PGP) which is expensive since the key must be encrypted individually for each member. Moreover, this method is insecure since compromise of just one member's long-term secret (PGP private key) exposes all previous session keys, and, hence, all prior group communication. (A more secure approach would avoid this problem by providing *Perfect*

Forward Secrecy, i.e., it would preclude distribution of keys encrypted under long-term secrets.)

6.8 Summary

This chapter represents only the tip of the proverbial iceberg. Much work remains to be done in constructing a comprehensive architecture and implementation for secure and reliable group communication.

As the immediate next step, secure handling of cascaded group events must be implemented as sketched out in Section 6.5. Also, more experimentation is needed to better assess the impact of security on group communication. In the short term, we need to consider other important and more specialized group security services that can be built on top of the basic services, i.e., distributed key management and data privacy/integrity. These include:

- intra-group member authentication
- secure communication with non-members (authentication, privacy, integrity)
- group integrity (how to authenticate the entire membership at once)
- group/member anonymity
- membership non-repudiation

There are also several fundamental, long-term research topics that this work has not addressed. Dynamic peer groups present interesting policy considerations that traditional two-party communication does not. New policy constraints might include: altering group membership; policy contingent on current membership; time based group admission constraints. Group certification is also an open research problem, including such issues as: how to issue, manage and revoke certificates for constantly changing groups. Finally, there is the need to extend peer group security to two-tiered groups composed of small number of senders and a comparatively large number of receivers.

Appendix A: Centralized Key Distribution protocol

The Centralized Key Distribution (CKD) protocol is a simple group key distribution scheme. It provides the same level of security as CLIQUES, as far as key independence, key confirmation, perfect forward secrecy and resistance to known key attacks. However, it is not contributory as the group secret is always generated by one member, namely, the current group controller.[7] Following each membership change, the current controller generates the secret and distributes it to the group in a secure manner. Unlike CLIQUES, the controller is always the oldest member.

Regardless of the group operation, the CKD key distribution protocol consists of two phases:

[7]We use the term *current* to mean that, even in the CKD protocol suite, a controller can fail or be partitioned out thus causing the controller role to be reassigned to the oldest surviving member.

<u>Protocol CKD:</u> Let (x_1, α^{x_1}) and $(x_{n+1}, \alpha^{x_{n+1}})$ be the secret and public-keys of M_1, the group controller, and M_{n+1} respectively. Let $K_{1n+1} = \alpha^{x_1 x_{n+1}}$ mod p. Assume that the group has n members, and that M_{n+1} wants to join the group. The protocol runs as follows: **Round 1:** M_1 selects random $r_1 modq$ (this selection is performed only once), $M_1 \longrightarrow M_{n+1} : \alpha^{r_1} modp$ **Round 2:** M_{n+1} selects random $r_{n+1} modq$, $M_1 \longleftarrow M_{n+1} : \alpha^{r_{n+1} K_{1n+1}} modp$ **Round 3:** M_1 selects a random group secret Ks and computes $M_1 \longrightarrow M_i : Ks^{\alpha^{r_1 r_i}} modp \forall i \in [2..n+1]$

Table 6.5: CKD protocol

1. Each group member and the controller agree on a key using authenticated two-party Diffie-Hellman.
   This key does not need to change as long as both users remain in the group. If the controller leaves the group, the new controller has to perform this operation with every member. On the other hand, if a regular member leaves, the controller simply discards this key.

2. The group controller unilaterally generates and distributes the group secret.

Appendix B: Multicast Communication Layers

Multicast communication layers usually support process group communication. Groups are conveniently identified by a name (string) that is selected by the user, such that messages are addressed to the entire group by specifying the group name. Using the group abstraction, the user is relieved from identifying the targets of messages explicitly, and from finding the network routes to them.

Using TCP for group communication is possible but rather complicated. First, in order to reach every group member, messages should be repeated over different channels and a socket should be installed between each two processes. Second, TCP does not report any membership changes. Thus, for instance, two communicating users are forced to use timeout mechanisms to be aware of any connection failures.

On the contrary, multicast communication layers take advantage of multicast technology, report any membership changes, and, generally, support the following ordering/delivery guarantees for messages:

- *Unreliable.* The message may not be delivered.
- *Reliable.* The message will be delivered but no assumptions are made regarding relative message arrival (no ordering).
- *Fifo.* Reliable messages ordered such that all messages from a member are delivered in the order in which they were sent.
- *Causal.* Reliable messages in FIFO order such that if a message A is received and then B is sent, it is guaranteed that all group members will receive A before B.
- *Agreed.* Reliable messages that are totally ordered, i.e., all group members receive all messages in the same order.
- *Safe.* Agreed messages such that delivery of a message occurs after all the machines in the network have received a copy of it.

Bibliography

[1] Y. Amir, "Replication using group communication over a partitioned network", Ph. Thesis from the Institute of Computer Science, The Hebrew University of Jerusalem, 1995.

[2] Y. Amir, G. Ateniese, D. Hasse, Y. Kim, C. Nita-Rotaru, T. Schlossnagle, J. Schultz, J. Stanton and G. Tsudik, "Secure Group Communication in Asynchronous Networks with Failures: Integration and Experiments", accepted for publication in *2000 International Conference on Distributed Computing Systems*. Also appeared as IBM Research Report.

[3] Y. Amir, D. Dolev, S. Kramer, and D. Malki. A communication sub-system for high availability. In *22nd Annual International Symposium on Fault-Tolerant Computing (FTCS)*, 1992.

[4] Y. Amir, D. Dolev, S. Kramer, and D. Malki, "Transis: A communication subsystem for high-availability," in *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*, pp. 76–84, IEEE Computer Society Press, Los Alamitos, CA, 1992.

[5] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. Agarwal, and P. Ciarfella, "The totem single-ring ordering and membership protocol," *ACM Transactions on Computer Systems*, vol. 13, pp. 311–342, November 1995.

[6] Y. Amir and J. Stanton. The Spread Wide Area Group Communication System. Technical Report CNDS 98-4, The Johns Hopkins University. Avalaible at `http://www.cnds.jhu.edu/projects/commedia/spread/docspread.html`

[7] R. Anderson and S. Vaudenay. Minding your p's and q's. In *Advances in Cryptology – Asiacrypt'96*, 1996.

[8] T. Anker, G. V. Chockler, D. Dolev, and I. Keidar, "Scalable group membership services for novel applications," in *Proceedings of the workshop on Networks in Distributed Computing* (M. Mavronicolas, M. Merritt, and N. Shavit, eds.), DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1998.

[9] Giuseppe Ateniese, "Efficient Verifiable Encryption (and Fair Exchange) of Digital Signatures". In *6th ACM Conference on Computer and Communications Security (ACM CCS'99).*

[10] G. Ateniese, O. Chevassut, D. Hasse, Y. Kim, and G. Tsudik, "The design of a group key agreement API," IBM Research Report. *to appear in DARPA DISCEX'2000.*

[11] G. Ateniese, D. Hasse, Y. Kim, and Gene Tsudik, "CLIQUES: Secure Dynamic Peer Group Key Agreement Protocol", Internet-Draft, Expires: Jan, 2000

[12] G. Ateniese, M. Joye, J. Camenisch, and G. Tsudik, "A New Digital Signature Scheme and its Application to a Practical and Provably Secure Coalition-resistant Group Signature Scheme", submitted to Eurocrypt99, November 1999.

[13] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *ACM Symposium on Computer and Communication Security*, November 1998.

[14] G. Ateniese, M. Steiner, and G. Tsudik, "New multi-party authentication services and key agreement protocols," in *IEEE Journal of Selected Areas in Communication*, 1999. Accepted for publication, to appear in early 2000.

[15] G. Ateniese and G. Tsudik, *Group signatures à la carte*, Tenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '99), January 1999. To appear.

[16] ———, *Some open issues and new directions in group signatures*, Financial Cryptography (FC '99) (M. Franklin, ed.), Lecture Notes in Computer Science, Springer-Verlag, 1999. To appear.

[17] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology – EUROCRYPT '97*, vol. 1233 of *LNCS*, pp. 480–494, Springer-Verlag, 1997.

[18] C. Becker and U. Wille. Communication complexity of group key distribution. In *ACM Conference on Computer and Communication Security*, November 1998.

[19] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *ACM Symposium on Theory of Computing*, 1998.

[20] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communication Security*, pages 62–73, ACM Press, 1993.

[21] ______. Entity authentication and key distribution. In *Advances in Cryptology – CRYPTO*, 1993.

[22] K. P. Birman and T. Joseph, "Exploiting virtual synchrony in distributed systems," in *11th Annual Symposium on Operating Systems Principles*, pp. 123–138, November 1987.

[23] K. P. Birman and R. V. Renesse, *Reliable Distributed Computing with the Isis Toolkit.* IEEE Computer Society Press, March 1994.

[24] D. Boneh. The decision Diffie-Hellman problem. In *Algorithmic Number Theory (ANTS-III)*, vol. 1423 of *LNCS*, pp. 48–63, Springer-Verlag, 1998.

[25] C. Boyd. Digital Multi-signatures. Cryptography and Coding (H.J.Beker and F.C.Piper Eds.), Oxford University Press, 1989, pp 241-246..

[26] S. Brands. An efficient off-line electronic cash system based on the representation problem. Technical Report CS-R9323, Centrum voor Wiskunde en Informatica, April 1993.

[27] M. Burmester. On the risk of opening distributed keys. In *Advances in Cryptology – CRYPTO*, 1994.

[28] M. Burmester and Y. Desmedt. Towards practical proven secure authenticated key distribution. In *ACM Conference on Computer and Communication Security*, 1993.

[29] ______. A secure and efficient conference key distribution system. In *Advances in Cryptology – EUROCRYPT*, 1994.

[30] ______ Efficient and secure conference key distribution. In *Cambridge Workshop on Security Protocols*, volume 1189 of *Lecture Notes in Computer Science*, pages 119–129. Springer-Verlag, Berlin Germany, April 1996.

[31] Jan Camenisch, *Efficient and generalized group signatures*, Advances in Cryptology — EUROCRYPT '97 (W. Fumy, ed.), Lecture Notes in Computer Science, vol. 1233, Springer-Verlag, 1997, pp. 465–479.

[32] ______, *Group signature schemes and payment systems based on the discrete logarithm problem*, Ph.D. thesis, ETH Zürich, Zurich, February 1998, Published in ETH Series in Information Security and Cryptography, vol. 2, Hartung-Gorre, 1998.

[33] Jan Camenisch and Markus Michels, *A group signature scheme with improved efficiency*, Advances in Cryptology — ASIACRYPT '98 (K. Ohta and D. Pei, eds.), Lecture Notes in Computer Science, vol. 1514, Springer-Verlag, 1998, pp. 160–174.

[34] ———, *A group signature scheme based on an RSA-variant*, Tech. Report RS-98-27, BRICS, Aarhus, November 1998. An earlier version appears in [33].

[35] ———, Proving in zero-knowledge that a number is the product of two safe primes. In *Advances in Cryptology – EUROCRYPT '99*, *Lecture Notes in Computer Science*, Springer-Verlag. To appear, 1999.

[36] ———, Separability and Efficiency for Generic Group Signature Schemes. In *Advances in Cryptology – Crypto '99*, to appear, 1999.

[37] Jan Camenisch and Markus Stadler, *Efficient group signature schemes for large groups*, Advances in Cryptology — CRYPTO '97 (B.S. Kaliski Jr., ed.), Lecture Notes in Computer Science, vol. 1294, Springer-Verlag, 1997, pp. 410–424.

[38] R. Canetti. Studies in secure multiparty computation and applications. PhD Thesis, Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, May 1995.

[39] B. P. Ran Canetti, A taxonomy of multicast security issues. draft-irtf-smug-taxonomy-00.txt, April 1999.

[40] A. Chan, Y. Frankel and Y. Tsiounis. Easy come – easy go divisible cash. In *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 561–575, Springer-Verlag, 1998. Updated and corrected version available as GTE Technical Report.

[41] D. Chaum, C. Crepeau, and I. Damgaard. Multiparty unconditional secure protocols. In *ACM Symposium on Theory of Computing*, 1988.

[42] D. Chaum and T. Pedersen. Wallet databases with observers. In *Advances in Cryptology – Crypto '92*, pages 89-105, 1992.

[43] David Chaum and Eugène van Heijst, *Group signatures*, Advances in Cryptology — EUROCRYPT '91 (D.W. Davies, ed.), Lecture Notes in Computer Science, vol. 547, Springer-Verlag, 1991, pp. 257–265.

[44] Lidong Chen and Torben P. Pedersen, *New group signature schemes*, Advances in Cryptology — EUROCRYPT '94 (A. De Santis, ed.), Lecture Notes in Computer Science, vol. 950, Springer-Verlag, 1995, pp. 171–181.

[45] ———, *On the efficiency of group signatures providing information-theoretic anonymity*, Advances in Cryptology — EUROCRYPT '95 (L.C. Guillou and J.-J. Quisquater, eds.), Lecture Notes in Computer Science, vol. 921, Springer-Verlag, 1995, pp. 39–49.

[46] R. Cramer and V. Shoup. Signature schemes based on the Strong RSA Assumption. To appear in *ACM Conference on Computer and Communication Security*, 1999.

[47] W. Dai, "Crypto++," May 1999. http://www.eskimo.com/~weidai/cryptlib.html.

[48] Yvo Desmedt, *Threshold cryptosystems*, Advances in Cryptology – AUSCRYPT '92 (J. Seberry and Y. Zheng, eds.), Lecture Notes in Computer Science, vol. 718, Springer-Verlag, 1993, pp. 3–14.

[49] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6): 644–654, 1976.

[50] A. Fekete, N. Lynch and A. Shvartsman, "Specifying and using a partionable group communication service," in *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, (Santa Barbara, CA), pp. 53–62, August 1997.

[51] A. Fiat and M. Naor. Broadcast Encryption In *Advances in Cryptology – CRYPTO*, 1993.

[52] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Springer-Verlag, 1987.

[53] Yair Frankel, Yiannis Tsiounis, and Moti Yung, *"indirect disclosure proofs": Achieving efficient fair off-line e-cash*, Advances in Cryptology — ASIACRYPT '96 (K. Kim and T. Matsumoto, eds.), Lecture Notes in Computer Science, vol. 1163, Springer-Verlag, 1996, pp. 286–300.

[54] E. Fujisaki and T. Okamoto. Statistical Zero Knowledge Protocols to Prove Modular Polynomial Relations. In *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 16–30, Springer-Verlag, 1997.

[55] ______. A practical and provably secure scheme for publicly verifiable secret sharing and its applications. In *Advances in Cryptology – EUROCRYPT '98*, vol. 1403 of *LNCS*, pp. 32–46, Springer-Verlag, 1998.

[56] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology – EUROCRYPT '99*, *LNCS*, Springer-Verlag, 1999.

[57] Marc Girault, *Self-certified public-keys*, Advances in Cryptology – EUROCRYPT '91 (D.W. Davies, ed.), Lecture Notes in Computer Science, vol. 547, Springer-Verlag, 1991, pp. 491–497.

[58] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[59] H. Harney and C. Muckenhirn, "Group key management protocol (GKMP) specification," Request for Comments (Experimental) 2093, Internet Engineering Task Force, July 1997.

[60] K. E. Hickman and T. Elgamal. The SSL protocol, RFC draft, Netscape Communications Corp., June 1995. Version 3.0, expires 12/95.

[61] C. Hoon Lim and P. Joong Lee. A key recovery attack on discrete log-based schemes using a prime order subgroup. In *Advances in Cryptology – CRYPTO*, 1997.

[62] T. Hudson and E. Young, "SSLeay," Dec. 1998. http://www.ssleay.org/.

[63] Marc Joye, Narn-Yih Lee, and Tzonelih Hwang, *On the security of the Lee-Chang group signature scheme and its derivatives*, Tech. Report TR-99-7, Tamkang LCIS, Tamsui, May 1999.

[64] S. T. Kent, "Internet privacy enhanced mail," *Communications of the ACM*, vol. 36, pp. 48–60, Aug. 1993.

[65] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "The securering protocols for securing group communication," in *Proceedings of the IEEE 31st Hawaii International Conference on System Sciences*, vol. 3, (Kona, Hawaii), pp. 317–326, January 1998.

[66] J. Kilian and E. Petrank. Identity escrow. In *Advances in Cryptology – CRYPTO '98*, vol. 1642 of *LNCS*, pp. 169–185, Springer-Verlag, 1998.

[67] Seungjoo Kim, Sangjoon Park, and Dongho Won, *Convertible group signatures*, Advances in Cryptology — ASIACRYPT '96 (K. Kim and T. Matsumoto, eds.), Lecture Notes in Computer Science, vol. 1163, Springer-Verlag, 1996, pp. 311–321.

[68] N. Koblitz. *A Course in Number Theory and Cryptography.* Springer-Verlag, Berlin Germany, Berlin, 1987.

[69] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: keyed-hashing for message authentication," Request for Comments (Informational) 2104, Internet Engineering Task Force, Feb. 1997.

[70] Narn-Yih Lee, *Personal communication*, May 19, 1999.

[71] Wei-Bin Lee and Chin-Chen Chang, *Efficient group signature scheme based on the discrete logarithm*, IEE Proc. Comput. Digit. Tech. **145** (1998), no. 1, 15–18.

[72] Chae Hoon Lim and Pil Joong Lee, *Remarks on convertible group signatures of Asiacrypt '96*, Electronics Letters **33** (1997), no. 5, 383–384.

[73] N. Lynch. Distributed algorithms. Morgan Kaufmann, San Francisco 1996.

[74] Anna Lysyanskaya and Zulfikar Ramzan, *Group blind signatures: A scalable solution to electronic cash*, Financial Cryptography (FC '98) (R. Hirschfeld, ed.), Lecture Notes in Computer Science, vol. 1465, Springer-Verlag, 1998, pp. 184–197.

[75] Wenbo Mao and Chae Hoon Lim, *Cryptanalysis in prime order subgroups of* $\mathbb{Z}_n$, Advances in Cryptology — ASIACRYPT '98 (K. Ohta and D. Pei, eds.), Lecture Notes in Computer Science, vol. 1514, Springer-Verlag, 1998, pp. 214–226.

[76] Ueli M. Maurer and Yacov Yacobi, *Non-interactive public-key cryptography*, Advances in Cryptology – EUROCRYPT '91 (D.W. Davies, ed.), Lecture Notes in Computer Science, vol. 547, Springer-Verlag, 1991, pp. 498–507.

[77] Markus Michels, *Comments on some group signature schemes*, Tech. Report TR-96-3-D, University of Technology Chemnitz-Zwickau, Chemnitz, November 1996.

[78] T. Matsumoto, Y. Takashima, and H. Imai. On seeking smart public-key-distribution systems. *Transactions of the IECE*, E69, 1986.

[79] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of applied cryptography.* CRC Press series on discrete mathematics and its applications. CRC Press, 1996. ISBN 0-8493-8523-7.

[80] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal, "Extended virtual synchrony," in *Proceedings of the IEEE 14th International Conference on Distributed Computing Systems*, pp. 56–65, IEEE Computer Society Press, Los Alamitos, CA, June 1994.

[81] L. Moser, P. Melliar-Smith, D. Agarwal, R. Budhia, and C. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. In *Communications of the ACM*, vol. 39, no. 4, 1996.

[82] P. Narasimhan, K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "Providing support for survivable corba applications with the immune system," in *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, (Austin, TX), pp. 507–516, May 1999.

[83] National Bureau of Standards, "Data Encryption Standard DES, Data Encryption Standard Model of Operation." Government Printing Office, Washington, D.C., 1980. Federal Information Processing Standards FIPS-81.

[84] NIST Computer Systems Laboratory. Digital signature standard (draft). FIPS PUB 186, May 1994.

[85] NIST Computer Systems Laboratory. Secure hash standard (draft). FIPS PUB 180-1, May 1994.

[86] OpenSSL Project team, "Openssl," May 1999. http://www.openssl.org/.

[87] Sangjoon Park, Seungjoo Kim, and Dongho Won, *ID-based group signature*, Electronics Letters **33** (1997), no. 19, 1616–1617.

[88] Sangjoon Park, Insook Lee, and Dongho Won, *A practical group signature*, 1995 Japan-Korea Joint Workshop on Information Security and Cryptography (JW-ISC '95), Japan, January 1995, pp. 127–133.

[89] Sangjoon Park and Dongho Won, *A practical identity-based group signature*, 1995 International Conference on Electronics, Informations and Communications (ICEIC '95), China, August 1995, pp. II–64–II–67.

[90] Holger Petersen, *How to convert any digital signature scheme into a group signature scheme*, Pre-proceedings of the 1997 Workshop on Security Protocols, Paris, April 7–9, 1997.

[91] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology – EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Springer-Verlag, 1996.

[92] G. Poupard and J. Stern. Security analysis of a practical "on the fly" authentication and signature generation. In *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 422–436, Springer-Verlag, 1998.

[93] R. V. Renesse, K. Birman, and S. Maffeis, "Horus: A flexible group communication system," *Communications of the ACM*, vol. 39, pp. 76–83, April 1996.

[94] O. Rodeh, K. Birman, M. Hayden, Z. Xiao, and D. Dolev, "Ensemble security," Tech. Rep. TR98-1703, Cornell University, Department of Computer Science, September 1998.

[95] RSA Laboratories. Rsaref2.0(tm): A cryptographic toolkit for privacy-enhanced mail. Jan. 1996. `http://www.rsa.com.`

[96] B. Schneier, "The blowfish encryption algorithm," *Dr. Dobb's Journal*, pp. 38–40, Apr. 1994.

[97] Adi Shamir, *Identity-based cryptosystems and signatures schemes*, Advances in Cryptology – Proceedings of CRYPTO 84 (G.R. Blakley and D. Chaum, eds.), Lecture Notes in Computer Science, vol. 196, Springer-Verlag, 1985, pp. 47–53.

[98] C.P. Schnorr. Efficient signature generation by smart-cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[99] J. Smith and F. Weingarten. Research challenges for the next generation internet, May 1997. Report from the Workshop on Research Directions for NGI.

[100] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In *Advances in Cryptology – CRYPTO*, 1988.

[101] J. G. Steiner, C. Neuman, and J. I. Schiller, "Kerberos: An authentication service for open network systems," in *Usenix Winter Conference*, pp. 191–202, Jan. 1988.

[102] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-hellman key distribution extended to groups," in *3nd ACM Conference on Computer and Communications Security*, pp. 31–37, ACM Press, Mar. 1996.

[103] ———, "CLIQUES: A new approach to group key agreement," in *IEEE International Conference on Distributed Computing Systems*, May 1998.

[104] Yuh-Min Tseng and Jinn-Ke Jan, *A novel ID-based group signature*, 1998 International Computer Symposium, Workshop on Cryptology and Information Security (T.L. Hwang and A.K. Lenstra, eds.), Tainan, December 17–19, 1998, pp. 159–164.

[105] ———, *Improved group signature scheme based on discrete logarithm problem*, Electronics Letters **35** (1999), no. 1, 37–38.

[106] ———, *A group signature scheme using self-certified public-keys*, Ninth National Conference on Information Security, May 1999, pp. 165–172.

[107] Chih-Hung Wang, Tzonelih Hwang, and Narn-Yih Lee, *Comments on two group signatures*, Information Processing Letters **69** (1999), 95–97.

[108] B. Whetten, T. Montgomery, and S. Kaplan, "A high performance totally ordered multicast protocol," in *Theory and Practice in Distributed Systems, International Workshop*, Lecture Notes in Computer Science, p. 938, September 1994.

[109] P. Zimmerman, *The Official PGP User's Guide*. prz@acm.org, 1994. The MIT Press In press. More in http://www.pegasus.esprit.ec.org/people/arne/pgp.html.