UNIVERSITÀ DEGLI STUDI DI BRESCIA
FACOLTÀ DI INGEGNERIA
LAUREA SPECIALISTICA IN INGEGNERIA INFORMATICA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
Elettronica - Informatica - Telecomunicazioni e Automatica

Storage Peer-to-Peer Sicuro: Tecniche ed Architetture

Candidato
Diego Ferri
Matricola
70184

Relatore
Prof. Luca Salgarelli
Correlatore
Ing. Francesco Gringoli

ANNO ACCADEMICO 2009/2010

Alle nonne,
che questa volta,
non sono più
qui con me

Abstract

The aim of our work is to show the feasibility of techniques that allow to achieve simultaneously privacy and security in large distributed storage systems running in a P2P environment. In our case, we define privacy as the possibility for users to take part in the services of the network with their identity anonymized. At the same time, we want to add an additional security layer for all network peers through a group authentication mechanism, meant as basic from of trust among them. This apparent contradiction between anonymization and identification forms the core of the problem that we will analyze, together with the requirement that our solution be scalable to a large number of users.

Our suggestion is to accomplish these goals through the use of *group signatures*, a modern cryptographic primitive that, in name, assures all three properties at the same time.

Initially, we will begin with a study of the properties of group signatures with a particular focus on two instances. Our original contribution will be the introduction of a model to study the overhead produced by group signatures in a system with a large number of users. We will then apply such model to a real scenario of a distributed storage system in a P2P environment: the PERIMETER storage system. Later on, we will demonstrate with the joined evaluation of both models that protecting the system with group signatures is not only feasible but even effective, with respect to our goals.

At the same time we will fill a gap left by the cryptographic community, developing a software framework for the use of group signatures and integrating it into already established cryptographic architectures.

Sommario

Il nostro lavoro di tesi mira a dimostrare la fattibilità di tecniche che permettano di ottenere contemporaneamente privacy e sicurezza in grandi sistemi di storage distribuito in ambiente P2P. Nel nostro caso, decliniamo l'idea di privacy come la possibilità per gli utenti di partecipare ai servizi della rete anonimizzando la propria identità. Contemporaneamente vogliamo garantire un layer aggiuntivo di sicurezza a tutti i peer della rete attraverso un meccanismo di identificazione di gruppo, inteso in questo senso come una basilare forma di trust tra di essi. Questa tensione tra il requisito di anonimizzazione e di identificazione costituisce il nocciolo del problema che andiamo ad analizzare, insieme con il requisito che la soluzione sia scalabile ad un grande numero di utenti.

La nostra proposta è di raggiungere questi obiettivi attraverso l'utilizzo di moderne primitive crittografiche: le firme di gruppo, in inglese *group signatures*, che sulla carta promettono tutte e tre le proprietà.

Inizialmente studieremo le proprietà delle firme di gruppo concentrandoci, in particolare, su due particolari esempi. Il nostro contributo originale sarà proporre dapprima un modello per studiare l'overhead introdotto dalle firme di gruppo applicate ad un sistema con un gran numero di utenti. Applicheremo poi tale modello a uno scenario reale di sistema di storage distribuito in ambiente P2P: il sistema di storage del progetto PERIMETER. A seguire si dimosterà con la valutazione congiunta dei due modelli che proteggere il sistema con le firme di gruppo non è solo fattibile ma anche efficace, in quanto raggiunge tutti gli obiettivi iniziali.

Parallelamente colmeremo un vuoto lasciato dalla comunità crittografica, portando avanti e descrivendo lo sviluppo di un framework software per l'utilizzo delle firme di gruppo e l'integrazione in architetture già consolidate.

Ringraziamenti

Voglio ringraziare per primo il Prof. Luca Salgarelli, mio relatore, per la disponibilità e l'incoraggiamento. Grazie per aver creduto vivamente in questo lavoro di tesi e per avermi dato la possibilità di misurarmi con un'esperienza di vita all'estero. Spero di fare tesoro anche solo di una piccola parte dei consigli che mi ha donato. Grazie anche all'Ing. Francesco Gringoli, mio correlatore, e al Dr. Fikret Sivrikaya, mio tutor al DAI-Labor presso la Technische Universität Berlin, per il supporto fornitomi.

Un grosso ringraziamento ai miei genitori che mi hanno sempre incoraggiato, sostenuto (anche economicamente!) e assecondato nella scelte riguardanti la mia formazione. Sono sicuro di aver imparato da loro a portare avanti le mie idee e a confrontarmi in modo chiaro ed onesto con gli altri.

Un grazie particolare alle nonne, che purtroppo ci hanno lasciato entrambe durante il percorso della laurea specialistica. Spero di custodire sempre nel cuore il loro insegnamento che la vita va pensata con progettualità, ma vissuta nella semplicità e nella gioia dei piccoli gesti quotidiani. La massima "*Un passo alla volta, figlio mio*", mi risuona spesso nelle orecchie.

Un sentito grazie anche ad Elena, da cui tanto ho imparato, ma tanto, sono sicuro, ho ancora da imparare: il suo mix di creatività, razionalità e modestia è una ricetta potente.

Un grosso grazie ad Anna per incoraggiarmi, sostenermi e starmi vicino ogni giorno, specialmente in questo ultimo periodo di tesi all'estero. È stato fondamentale per me sapere di avere la sua approvazione e partecipazione a quello che andavo scoprendo.

Un grazie a Lino per essere sempre stato un amico sincero e una guida preziosa che con il suo stile "scomodo" mi ha donato la possibilità di mettermi in discussione. Se oggi credo di essere un pò meno "quadrato", lo devo a lui.

Un grazie congiunto a Don Giovanni e a tutto il gruppo di animatori, perchè certe frasi, certe esperienze e certe scelte sono sicuro che hanno lasciato e lasciano un segno indelebile.

Un grazie a tutta la mia compagnia di amici, il tempo passa, le situazioni cambiano, qualcuno va e qualcuno torna. Ma è sempre bello avere certezze nella vita: Carmine, Michele, Domenico, Marsel, Luca, Enrico, Andrea, Matteo (tutti e 3!) sono una di quelle.

Un grazie a tutto il gruppo dell'università in tutte le sue forme: informatici in primis, ma anche di telecomunicazioni e addirittura elettronici! Imparare e migliorarsi sono valori fondamentali nella vita ma senza relazioni che diano senso a quello che si fa, temo sarebbero sterili e fini a sè stessi. Grazie per aver reso speciale tutti gli anni spesi tra le mura di Ingegneria. Un grazie particolare ad Andrea, Manuel, Paolo e Madda per un'amicizia che è sicuramente andata

ben oltre i banchi accademici.

Ultimi, ma non certo per importanza, i miei più stretti compagni dell'avventura berlinese: Ruien, Peter e Luisa. Sono arrivato sperduto nella grande città, ma una volta incontrati mi sono subito sentito a casa. Ho cercato dei coinquilini, ho trovato molto di più. Sono sicuro che, anche se una pagina si è chiusa, non si tratta di un addio. Grazie per il tempo passato insieme. Remember: "Berlin calling". Danke für alles!

Berlino, 28 aprile 2010

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

1.1 Motivations

Privacy is acknowledged and protected as a fundamental and inviolable right by many legal orders, national and supranational, even if it is a concept hard to pinpoint clearly: it is often used as a blanket term with many dimensions, e.g. identity, location and personal data privacy, and meanings that are very context-dependent. Thus, in practice, it is frequently difficult to deploy means to protect user privacy. Many systems that we use everyday were not designed at all for privacy protection or, most of the times, it was added as an afterthought.

Traditionally, identification has been a disclose-all-or-nothing action. Just think, during our everyday life, how many times we are requested to identify ourselves, either digitally or not. More often than not together with a great deal of additional details, even if the only relevant information required is, for instance, the membership of a particular group. Imagine a big company where employees are asked to identify every time they move around the company site, to check if they have clearance to move to a new area (e.g. a lab or a department). Nowadays, it is a common solution to carry a personal badge for identification. Of course, this course of action works but has one critical drawback: an employee can be tracked to an unprecedented degree during the daily routine, although this can be unpolite at the best of times; sometimes even a criminal offence. The true identity of an employee could, and should, in this case be protected and hidden.

On the other side of the fence, the company is legitimately interested in the security of its own site. Security is another fundamental value for society and pertains the protection of people, structures, processes and systems. In this case security is obviously related to the protection of the company's site, especially with respect to outsiders. The company wants the confidence that a person roaming the building is without doubt a company's employee as a very basic, binary form of trust: one is in the group of insiders or he is not. Moreover, in case of abuse the compelling need to identify the offender arises. So, how to fulfill the needs of both parts?

It is clear enough from this example, that privacy and security are often considered two irreconcilable and opposite goals: increasing security usually means to hinder the aspiration to privacy. On the contrary, enhancing the

privacy of the people, many times can lead to poor security standards, such as in the case described above. In fact, it is believed that it is impossible to increase one without sacrificing the other and vice versa. The scenario already depicted, however, is only a very simple example where privacy and security intertwine together. We give in the following a more complex and realistic scenario.

Imagine the case of a distributed system where a large number of users wants to share feedback about the quality of mobile and wireless services present in an area, so that everyone will be able to choose the service that best suits their preferences and uses. Every user taking part in the system has the possibility from time to time to rate subjectively the quality of a service used and to distribute this feedback to other users upon request, together with the location where the measurement was taken. In case of abuse (e.g. tampering with the feedback repeatedly), a trusted authority is even able to recover a user's identity.

This peer-to-peer system is potentially a perfect way to share information otherwise hard to obtain and could be a killer application of the future. It raises, however, the concerns already discussed above. Is it possible to anonymize the identity of users, so that their location is not publicy available to everyone? Is it possible to do the same for feedbacks, so that network operators cannot penalize them in retaliation? Is it possible to trust feedbacks received by users whose identity is concealed? In this anonymized system, is it possible to trace identities in case of abuse? Is this system scalable to a large number of users?

1.2 Our contribution

In this work we will show that the idea of sacrificing security for privacy and vice versa is based on old assumptions and that a tradeoff among anonymity, trust, traceability and scalability is probably not necessary.

We have already discussed that these requirements seem to negate each other, but we show that thanks to the use of advanced cryptographic techniques, called *group signatures*, the goals are reachable. The main contribution of our work is to demostrate the feasibility of obtaining, at the same time, scalability, strong privacy and strong security properties in a large scale, mobile distributed system, like the one described above. In this context, privacy means identity anonimity and unlinkability of transactions. Security, in contrast, means group identification as a simple form of trust. In addition, we will be able to revoke anonymity in case of disputes or of abuses.

We will first provide in chapter 2 the general background and state of the art about group signatures, while in appendix B the number theory of interest to understand the group signature schemes described in chapters 3 and 4.

Then in chapter 5 we briefly describe the software development process of an experimental framework for group signatures.

Chapter 6 analyzes the performances and sketches a single peer and a system model for group signatures employed in a large distributed system. Finally, in chapter 7 we try to provide an original answer to the scenario described at the end of the previous section. We do this by making an analysis for the group signature model applying it to a distributed storage system running in a 3G mobile network.

Chapter 2

Background on Group Signatures

In this chapter we give a general overview of history, scope, advantages and disadvantages of group signatures, which are the building block of our work.

2.1 Overview

A *group signature scheme* is a digital signature scheme with enhanced privacy features. Partecipants in a group signature scheme are a set of *group members*, a *group manager* and possibly a group of non-member users. A member of the group can sign a message anonymously on behalf of the group, producing a *group signature*. Anyone with the group public key can then verify the validity of a group signature. The group manager, a trusted individual, is the only one that can setup a new group, admit new users into the group by issuing membership certificates and, in the case of a dispute, revoke anonymity of malicious users by *opening* signatures. Group signatures are anonymous and unlinkable, that is it is computionally hard to establish the identity of the signer and whether, or not, multiple signatures were produced by the same member.

2.2 Concepts

2.2.1 Actors

We define here the actors that take part in a group signature scheme:

Group Manager trusted individual with group administrarion and de-anonymization capabilities;

Group Member user with the capability of signing on behalf of the group;

Non-Member user not member of the group.

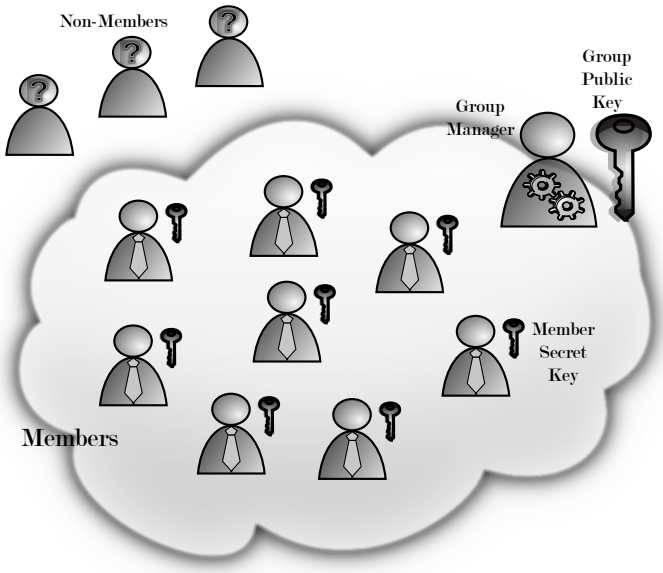

Figure 2.1: Group signature actors

2.2.2 Operations

A scheme is composed by a set of interoperating operations:

Setup generates the group keys on input the security parameters;

Join a protocol between the group manager and a user that results in the non-member to become a group member and to receive a membership proof;

Sign an algorithm that lets a member of the group produce a signature of a message;

Verify an algorithm to verify the validity of an alleged group signature;

Open a procedure that, for valid signatures, outputs the identity of the signer;

Revoke a procedure that allows the group manager to revoke membership of a group member. This procedure is present only in some schemes.

2.2.3 Properties

We have said that a group signature scheme is a digital signature scheme with privacy enhancing properties. Different schemes propose different properties but we would like to summarize here informally a base set of properties, described in [ACJT00].

Anonymity given a valid signature of some message, identifying the actual signer is computationally hard for everyone but the group manager;

Correctness signatures produced by a valid group member using *Sign* must always be accepted by *Verify*;

Unforgeability only group members are able to sign messages on behalf of the group;

Unlinkability deciding whether two different valid 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;

Non-Framing a group member is not responsible for signatures that he has not produced;

Traceability the group manager is always able to open a valid signature and identify the actual signer. This property can also be violated if a subset of group members, pooling together their secrets, can generate a valid group signature that cannot be opened by the group manager;

Coalition-Resistance a colluding set of group members, even if comprised of the entire group, cannot generate a valid signature that the group manager cannot link to one of the colluding group members.

More recently, these properties have been consensed in a more concise way and are used more formally as the base of the security model in [BMW03] (we point the reader to the original paper for the adversarial model and the actual proofs of security).

Full-Traceability without a member's secret it must be infeasible to create valid group signatures that frame a member, even if the group manager's secret key or an arbitrary number of member secret keys are compromised by an attacker;

Full-Anonymity it must be infeasible to distinguish signatures signed by different members, even if their secret keys are exposed, for signatures produced in the past or to get some clues on signatures produced in the future.

The properties above are only the base set. We can add other desiderable ones to the list:

Dynamic members may be added to the group even after the setup of the group itself;

Scalability the scheme must be suitable for use with large groups of users;

Practical Security regardless of the theoretical analysis, all the algorithms and protocols must be usable in practice (for instance, an exponential algorithm could still be fast enough to be pratical);

Provably-Secure it can be shown that defeating the scheme is as difficult as solving a well-known number theoretic problem (usually under cryptographic assumptions);

Revocation it should be possible to shrink the group size revoking membership to a group member.

2.2.4 Efficency

The efficency of a group signature scheme is usually measured on a small set of indicators:

- the size of group public key;
- the size of the group signature;
- the efficency of *Sign* and *Verify*;
- the efficency of *Setup*, *Join* and *Open*;
- the efficency of *Revoke*, when present.

2.3 Background

Group signatures are a relatively new advanced cryptographic tool, introduced in 1991 by work of [CvH91]. Like ordinary digital signatures, they allow a signer to demostrate knowledge of a secret with respect to a message. In addition, they provide the signer with anonymity and unlinkability of different signatures. Only the group manager, from now on GM, is able to "open" a signature, revoking anonymity and recovering the signer's identity.

It is interesting to note that a concept dual to group signatures (and in practice often the origin of the schemes) is identity escrow [KP98]. It can be regarded as a group identification scheme with revocable anonymity. In fact, any identity escrow scheme can be turned into a group signature scheme by applying the Fiat-Shamir heuristic [FS86] to the protocol for providing membership; the opposite is achieved by signing a random message and then proving the knowledge of a signature on the chosen message.

Chaum and van Heist proposed four static schemes, in which the addition of new members is possible only at the cost of reforming the group with a new *group public key*, $\mathcal{GPK}$, and new membership certificates. Security and efficiency problems were addressed by later work of [CP95] and [Cam97]. All these solutions, however, suffer from undesiderable drawbacks:

- the length of the $\mathcal{GPK}$ and/or the size of a signature depend on the size of the group so that they are not suited for large groups;
- to add new group members, it is necessary to modify at least the $\mathcal{GPK}$.

The scheme of [CS97] is a major breakthrough in the field. In fact, it proposes a scheme where all the following are independent of the number of group members:

- the length of the $\mathcal{GPK}$;

- the length of the signatures;
- the computational effort required for signing;
- the computational effort required for verifying.

In addition, there is no need to change the $\mathcal{GPK}$ every time a new member is added to the group. This effectively paves the way for dynamic groups of a, possibly, large set of members. The term dynamic anyway may be misleading, because no scheme of the time solved the problem of revocation of membership (e.g., due to expiration of the subscription to the group or due to persistent malicious behaviour of a member): the term monotonically growing has also been suggested instead.

Several new schemes, based on the same assumptions, with interesting improvements were then proposed by [Cam98], [CM98b], [CM98a] and [KY04]. Some have been subsequently broken or are inefficient or offer unproven security. Ateniese and Tsudik [AT99] pointed out some obstacles that stand in the way of real world applications of group signatures, such as coalition attacks and member deletion. Basically at this time, the issue of removing a member from the group was solved by periodically invalidating every member certificate and reissuing new ones for the members still allowed to sign.

The first practical, coalition-resistant and provable secure scheme is from Ateniese et al. [ACJT00], from here on simply referred as ACJT. This scheme has stood the test of time and for many years has been regarded as the state of the art.

Still, even with ACJT, the issue of shrinking group membership without incurring in massive computational costs, has not been solved. The solution to this problem is not merely an extension of the case of static groups because the dynamic case is more complex, bringing in the arena more requirements and issues. For instance, it is not possible to publish revoked member identities in a Member Revocation List, because of the anonymity feature. So how to prevent revoked member to produce valid signatures? Then, if the group manager reveals some secrets of a revoked member, past signatures are bound to be exposed or they rather retain the unlinkability feature? This design choices are sometimes referred as useful properties, sometimes as annoying drawbacks.

In fact, different scheme security definitions were proposed during the years, often with unformalized, ambigous, overlapping or even not agreed definitions. Bellare et al. with [BMW03] formalized and reduced the actual desiderable properties of a secure static group scheme to only two: full-anonymity and full-traceability. They even drew some criticism on the authors of [ACJT00] who claim their scheme secure, but without defining an attack model to the security of the scheme.

After ACJT, membership revocation received a good deal of attention. Bresson et al. in [BS01] presented a mechanism based on a member revocation list but where the length of the signature is linear in the number of revoked members. Song in [Son01] proposed two methods for revocation and forward security both using times periods. Ateniese et al. [AST02] work is based pretty much on the same basis: both suffer of inefficiency because the verification task is linear in the number of excluded group members.

The first step ahead on this issue was Camenisch and Lysyanskaya's work [CL02]. They put forward a dynamic accumulator based scheme. An accumula-

tor is a mathematical tool that accumulates primes into a short value and allows to dynamically add and delete inputs. Using such an accumulator, applied to the revocation of membership, does not alter the computational complexity which is still the same of the underlaying ACJT scheme, increased by only a constant factor, less than two. A signer, to prove membership, has then to prove that his own certificate has not been accumulated into the accumulator. There exist disavantages: the members must perform local computation to update their witness which is linear in the number of changes that have taken place from the last update and must frequently look at the public key as verifiers. The *GM* must update the group public key every time a member enters (only in the basic scheme) o leaves the group. It should come to no surprise that a modified version of ACJT together with the CL signature scheme extension has been employed as the building block for the TPM, Trusted Platform Module former Palladium, in the so called DAA (Direct Anonymous Attestation) scheme [BCC04]. [TX03] is more efficient than [CL02] but needs to update the group public key both when members join and leave the group and it's not strictly ACJT related.

To satisfy the requirements of Bellare's security model, it is impossible to revoke a group member except that all valid group members can somehow adjust the signing parameters or procedure, which may not always be feasible or efficient in practice. For the purpose of efficient revocation, many schemes [CG04, AST02, TX03, BS04] have adopted the so-called "verifier-local revocation" (VLR) technique, in which verifiers adjust their local verification parameters to recognize corrupted group members, and group members do not need to change the signing procedures at all. The scheme of Camenisch and Groth [CG04], from now on CG, is the first to support VLR together with a relaxed version of Bellare et al. security definitions. This scheme is based on the same assumptions of [ACJT00, CL02] but it offers revocation and it is at least one order of magnitude more efficient than both. Later on, other schemes [Ge04, ZW08] followed in the wake of CG but without introducing significant improvements.

At the same time of Camenisch and Groth's work, Boneh, Boyen, and Shacham [BBS04] as well as Camenisch and Lysyanskaya [CL04] presented group signature schemes based on bilinear maps. While these schemes produce shorter signatures, they are more computationally intensive [HP06]. In addition, they are based on very new and less studied number theoretic assumptions.

Chapter 3

Ateniese, Camenisch, Joye, Tsudik Scheme

This chapter provides an overview of Ateniese, Camenisch, Joye and Tsudik group signature scheme [ACJT00]. The security of the system is based on the SRSA assumption (see B.1.1) and on the DDH assumption (see B.2.1). The scheme is an improvement of [CM98b, CS97].

The scheme offers the first set of properties seen in section 2.2.3. In addition, it is provably secure, practical, scalable and dynamic, however, as seen in section 2.3, it does not support member revocation.

3.1 Definitions

In this section, we briefly report definitions useful for the understanding of the chapter.

Token	Description
GM	Group manager
m_i	Member of the group
$\mathcal{M} = \{m_1, \dots m_n\}$	Set of group members
$\mathcal{GPK}$	Group public key
$\mathcal{GSK}$	Group secret key
$\mathcal{MSK}$	Member secret key
Msg	Message to be signed/verified/opened
$Signature$	Group signature

Table 3.1: Group signature definitions

3.2 Procedures

In this section we review more formally the operations already entered in 2.2.2. The scheme supports only the following operations:

`SETUP` on input the security parameters, this algorithm outputs the GPK and the GSK. The $\mathcal{GPK}$, is then made publicly available even to non-members;

`JOIN` a protocol between the GM and a user outside the group that results in the user becoming a new group member, m_i. The member's output is a membership certificate with a $\mathcal{MSK}$;

`SIGN` a probabilistic algorithm that on input $\mathcal{GPK}$, a membership certificate, a membership secret and a message Msg outputs a group signature $Signature$ of Msg;

`VERIFY` a deterministic algorithm for establishing the validity of an alleged $Signature$ of Msg with respect to a $\mathcal{GPK}$;

`OPEN` an algorithm that, given a Msg, a valid $Signature$ on it, a $\mathcal{GPK}$ and a $\mathcal{GSK}$, determines the identity m_i of the signer.

3.3 Security Parameters

We report in table 3.2 the system parameters of [ACJT00] as described in the paper and the suggested values found in [NSN04, Ngu05][1].

The parameters have the following constraints among them:

$$\begin{aligned}
&\lambda_1 > \epsilon(\lambda_2 + k) + 2 \\
&\lambda_2 > 4l_p \\
&\gamma_1 > \epsilon(\gamma_2 + k) + 2 \\
&\gamma_2 > \lambda_1 + 2.
\end{aligned}$$

3.4 Scheme

In this section we describe the actual operations of the group signature scheme. Usually, as noted in [CM98b, CM98a, CS97, CG04] it can observed that there are two non-group signature schemes at work. The first is for issuing certificates of membership, the second is for the actual group signing/verifying. The second scheme is based on a proof of knowledge of a membership certificate through the Fiat-Shamir heuristic. The efficiency of the signature scheme is therefore strictly linked to the efficiency of the membership proof scheme. We refer the reader to appendix B for the mathematical concepts involved.

3.4.1 SETUP

This phase involves the GM only. It must be trusted, otherwise the security of the scheme is not guaranteed. The outputs of the phase are the $\mathcal{GPK}$ and the $\mathcal{GSK}$.

[1] We deduce from the proposed values that l_p in the constraints is not the bit length of the modulus but its number of digits ($n_{digits} = n_{bits} * log_{10}2$). This is not remarked in the original paper.

Parameter	Description	Values (bits)[a]
l_p	RSA modulus factor	512
ϵ	Tightness of the zero-knowledgeness proof	1.1
λ_1	Interval ranges for proving discrete logarithm knowledge	838
λ_2		600
γ_1		1102
γ_2		800
k	Digest length	160
$\mathcal{H}$	A collision-resistant hash function so that $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^k$	

[a] Values from [NSN04, Ngu05].

Table 3.2: ACJT scheme security parameters

Setup $(SecurityParameters) \rightarrow (\mathcal{GPK}, \mathcal{GSK})$

1. Set the integral ranges

$$\Lambda =]2^{\lambda_1} - 2^{\lambda_2}, 2^{\lambda_1} + 2^{\lambda_2}[$$
$$\Gamma =]2^{\gamma_1} - 2^{\gamma_2}, 2^{\gamma_1} + 2^{\gamma_2}[.$$

2. Generate a safe RSA modulus: select two random secret primes p' and q', each of l_p-bit length, so that $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes.
3. Set the modulus $n = pq$.
4. Choose random elements $a, a_0, g, h \in QR(n)$. $QR(n)$ is the subgroup of quadratic residues modulo n, i.e. the cyclic subgroup $QR(n)$ generated by an element of order $p'q'$, where the DDH assumption is conjectured to hold.
5. Choose a random secret element $x \in \mathbb{Z}^*_{p'q'}$.
6. Set $y = g^x \bmod n$.
7. Publish the $\mathcal{GPK} = (n, a, a_0, y, g, h)$, possibly signed by some kind of long term credentials, like digital signatures.
8. Save the $\mathcal{GSK} = (p', q', x)$.

We note that the component of the $\mathcal{GPK}$ must be verifiable to prevent framing attacks and that the GM needs to prove that the RSA modulus is a product of safe primes.

3.4.2 JOIN

This phase involves the GM and a non-group member that applies for membership. Both of them together generate the membership certificate using a zero-knowledge protocol. This phase needs to be secure, that is private and authentic. The outputs are a membership certificate, a membership secret and a new row in the membership table.

Join $(\mathcal{GPK}) \rightarrow (\mathcal{MSK})$

1. User generates a random integer $\bar{x} \in]0, 2^{\lambda_2}[$.
2. User generates a random integer $r \in]0, n^2[$.
3. User sends $C_1 = g^{\bar{x}} h^r \bmod n$ to the GM proving knowledge of the representation of C_1 with respect to the bases g and h.
4. GM checks that $C_1 \in QR(n)$. If it is false, the joining fails.
5. GM selects random $\alpha, \beta \in]0, 2^{\lambda_2}[$ and sends them to the user.
6. User computes $x = 2^{\lambda_1} + (\alpha \bar{x} + \beta \bmod 2^{\lambda_2})$ and $C_2 = a^x \bmod n$, so that x derives from C_1, α, β.
7. User sends $C_2 = a^x \bmod n$ to the the GM proving knowledge of (i) a discrete logarithm, (ii) equality of two discrete logarithm, (iii) a discrete logarithm lying in a given interval.
8. GM checks that $C_2 \in QR(n)$. If it is false, the joining fails.
9. GM generates a random prime $e \in \Gamma$ and computes $A = (C_2 a_0)^{e^{-1}} \bmod n^2$.
10. GM sends to the user the new membership certificate $[A, e]$.
11. User, now member m_i, verifies that certificate validity with $a^x a_0 \equiv A^e \bmod n$.
12. User sets $\mathcal{MSK} = (A, e, x)$.
13. GM updates the membership table with a new row $\{m_i, [A, e]\}$ together with the messages (signed by the user) exchanged during the protocol.

3.4.3 SIGN

This is the actual signing process. A valid member of the group, that is with a valid membership certificate and secret, may generate a group signature on a message. The output of the phase is the signature itself.

[2] e^{-1} is the modular inverse of e in the group of order $p'q'$, see [Ge04].

Sign $(\mathcal{GPK}, \mathcal{MSK}, Msg) \rightarrow (Signature)$

1. Generate a random integer $w \in \{0,1\}^{2l_p}$.
2. Compute

$$T_1 = Ay^w \bmod n$$
$$T_2 = g^w \bmod n$$
$$T_3 = g^e h^w \bmod n.$$

3. Choose

$$r_1 \in \pm\{0,1\}^{\epsilon(\gamma_2+k)}$$
$$r_2 \in \pm\{0,1\}^{\epsilon(\lambda_2+k)}$$
$$r_3 \in \pm\{0,1\}^{\epsilon(\lambda_1+2l_p+k+1)}$$
$$r_4 \in \pm\{0,1\}^{\epsilon(2l_p+k)}.$$

4. Compute

$$d_1 = T_1^{r_1}/(a^{r_2}y^{r_3}) \bmod n$$
$$d_2 = T_2^{r_1}/g^{r_3} \bmod n$$
$$d_3 = g^{r_4} \bmod n$$
$$d_4 = g^{r_1}h^{r_4} \bmod n.$$

5. Compute

$$c = \mathcal{H}(g \parallel h \parallel y \parallel a_0 \parallel a \parallel T_1 \parallel T_2 \parallel T_3 \parallel d_1 \parallel d_2 \parallel d_3 \parallel d_4 \parallel Msg).$$

6. Compute

$$s_1 = r_1 - c(e - 2^{\gamma_1})$$
$$s_2 = r_2 - c(x - 2^{\lambda_1})$$
$$s_3 = r_3 - cew$$
$$s_4 = r_4 - cw \text{ (all in } \mathbb{Z}).$$

7. Produce

$$Signature = (c, s_1, s_2, s_3, T_1, T_2, T_3).$$

3.4.4 VERIFY

This phase is still part of the signing protocol. It verifies if an alleged group signature is valid with respect to a given $\mathcal{GPK}$. The verifier could be a non-group member: he only needs the $\mathcal{GPK}$. The output of the procedure is a boolean value that is true if the signature is valid, false otherwise.

Verify $(\mathcal{GPK}, Msg, Signature) \rightarrow (ValidSignature?)$

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

2. Check if
$$s_1 \in \pm\{0,1\}^{\epsilon(\gamma_2 + k)+1}$$
$$s_2 \in \pm\{0,1\}^{\epsilon(\lambda_2 + k)+1}$$
$$s_3 \in \pm\{0,1\}^{\epsilon(\lambda_1 + 2l_p + k + 1)+1}$$
$$s_4 \in \pm\{0,1\}^{\epsilon(2l_p + k)+1}.$$

3. If the check is positive and $c = c'$ then output `true`, otherwise `false`.

3.4.5 OPEN

With this operation the GM revokes the anonymity feature and can retrieve the identity of the member who signed the group signature. The output of this phase is identity of the member.

Open $(\mathcal{GPK}, Msg, Signature, \mathcal{GSK}) \rightarrow (m_i)$

1. Check the validity of the group signature with the `VERIFY` procedure.
2. Compute $A = T_1 / T_2^x \bmod n$.
3. Recover the identity of the signer, m_i, searching for A in the membership table.
4. Prove that $\log_a y = \log_{T_2}(T_1 / A \bmod n)$.

Chapter 4

Camenisch, Groth Scheme

We provide in this chapter a description of Camenisch and Groth group signature scheme [CG04]. Its foundations are the same assumptions of [ACJT00], SRSA and DDH respectively (see B.1.1 and B.2.1).

The scheme is built on the second, more formal, set of properties seen in section 2.2.3, based on the work of [BMW03]. Once again, we point the reader to [CG04] for the actual proofs of these properties and the adversary model. In addition to the above properties, it is provably secure, practical, scalable, dynamic and, above all, supports member revocation. It improves ACJT, claiming to be more than one order of magnitude more efficient than pure ACJT.

In the original paper, three different versions of the scheme are described: one for static groups only, the others for dynamic groups. The dynamic case is more complex and so the authors offer different levels of anonymity revocation. The first revocation mechanism only revokes the possibility to produce new signatures to the revoked member, thus retaining the anonymity of past signatures. The second method, called *full revocation*, reveals the identity of the member of all signatures signed by a revoked member certificate. We will take into account only the full revocation variant.

4.1 Definitions

The definitions used in this chapter are the same as in section 3.1. We add only a concept, due to CG specificity:

Token	Description
MRL	Member revocation list

Table 4.1: Group signature definitions (CG specific)

4.2 Procedures

The procedures of this scheme are the same as the ones in section 3.2. However, since CG supports VLR, we have two procedures more:

`REVOKE` a procedure that allows the GM to revoke the signing capability of a group member. This is equivalent to remove the member from the group;

`FULL-REVOKE` a procedure that allows the GM to reveal a secret that "opens" every signature signed by a revoked member.

As we have already noted, the scheme supports the full revocation mode that allows the group manager to open every signature produced by a revoked group member. This feature clashes with the stronger full-anonymity requirement imposed by [BMW03]. Therefore for the group signature scheme with full revocation, Camenisch and Groth define a weaker security assumption where anonymity is granted if both $\mathcal{GSK}$ and $\mathcal{MSK}$ are not exposed. A byproduct of this choice is that a member can prove that he generated a specific signature; in [KY04] this feature is called *claiming*.

4.3 Security Parameters

In table 4.2 we review the meaning of the security parameters and the suggested values as they are described in [CG04] and [HP06]. For a better understanding we have to premise that the group signature scheme is based on two groups: one is QR_n, the group of quadratic residue modulo a safe prime RSA modulus, and the other is of order Q in $\mathbb{Z}_P^*$, where $\mathbb{Z}_P^*$ is the group of positive integers smaller than P and coprime with it and $Q|P-1$.

Parameter	Description	Values (bits)[a]
l_n	RSA modulus	2048
l_P	Order of the group $\mathbb{Z}_P^*$	2048
l_Q	Order of the group in $\mathbb{Z}_P^*$, where $Q\|P-1$	282
l_E	Prime number for member certificate	504
l_e	Number large enough to assign all members different numbers	60
l_s	Tightness of the zero-knowledgeness proof	60
l_c	Digest length	160
$\mathcal{H}$	A collision-resistant hash function so that $\mathcal{H}: \{0,1\}^* \rightarrow \{0,1\}^{l_c}$	

[a] Values from [CG04, HP06].

Table 4.2: CG scheme security parameters

The parameters have the following constraints among them:

$$\begin{aligned} l_c + l_e + l_s + 1 &< l_Q \\ l_Q + l_c + l_s + 1 &< l_E \\ l_E &< l_n/2. \end{aligned}$$

4.4 Scheme

In this section we describe the actual operations of the group signature scheme with full revocation.

4.4.1 SETUP

In the static group scenario, that we are not considering, the key generation algorithm is run by a trusted third party because it outputs, together with the group keys and system parameters, all membership certificates in advance. In the dynamic case this constraint is not needed since the secrets are generated jointly by both party, during the `JOIN`. Therefore this phase works like ACJT's `SETUP` and outputs the group keys and the system parameters.

Setup $(SecurityParameters) \rightarrow (\mathcal{GPK}, \mathcal{GSK})$

1. Generate a safe modulus: select two random primes p' and q', so that $p = 2p' + 1$, $q = 2q' + 1$ are safe primes.
2. Set the modulus $n = pq$ so that n is of length l_n bits.
3. Choose random elements $a, g, h \in QR(n)$, where $QR(n)$ is the subgroup of quadratic residue modulo n, i.e. the cyclic subgroup $QR(n)$ generated by an element of order $p'q'$.
4. Select two random primes Q and P, of l_Q and l_P-bit respectively, so that $Q|(P-1)$.
5. Let F as an element of order Q in $\mathbb{Z}_P^*$.
6. Choose at random $X_G, X_H \in \mathbb{Z}_Q$.
7. Set $G = F^{X_G}, H = F^{X_H} \bmod P$.
8. Choose random elements $w, f \in QR(n)$.
9. Publish the $\mathcal{GPK} = (n, a, g, h, Q, P, F, G, H, w, f)$.
10. Save the $\mathcal{GSK} = (p, q, X_G)$.

4.4.2 JOIN

As in ACJT, during this phase the GM and a would-be-member agree on a membership secret jointly generated. This protocol is more slender than ACJT's and only requires two rounds. The idea is that the member generates a secret by himself and then gets a Camenisch-Lysyanskaya signature [CL03] on it from the GM. In the exchange the element w_i is generated for revocation purposes. At the same time, for the full revocation to work we force the member to generate an element s_i: if this token is exposed by the group manager it is easy to link the member to every signature he has made.

Join $(\mathcal{GPK}) \rightarrow (\mathcal{MSK})$

1. User i generates a random integer $x_i \in \mathbb{Z}_Q$.
2. User computes $Y_i = G^{x_i} \bmod P$.
3. User forms a committment $g^{x_i} h^{r_i'} \bmod n$ to x_i, where r_i' is a random number in $\mathbb{Z}_n$.
4. User selects $s_i \in \mathbb{Z}_Q$.
5. User sends $Y_i, g^{x_i} h^{r_i'} \bmod n, s_i$ to the GM.
6. GM selects $e_i \in \{0,1\}^{l_e}$ such that $E_i = 2^{l_E} + e_i$ is prime.
7. GM computes $w_i = w^{E_i^{-1}} \bmod n$.
8. GM picks a random $r_i'' \in \mathbb{Z}_{e_i}$.
9. GM sets $y_i = (a f^{s_i} g^{x_i} h^{r_i' + r_i''})^{E_i^{-1}} \bmod n$[1].
10. GM sends back w_i, y_i, E_i, r_i'' to the user.
11. Member m_i sets $\mathcal{MSK} = (w_i, x_i, r_i = r_i' + r_i'', y_i, e_i)$.

4.4.3 SIGN

A valid member may sign on behalf of the group, producing a valid group signature. An important note is that a revoked member can still produce signatures under the old $\mathcal{GPK}$ and claim its validity the key was valid: an obvious countermeasure is the addition of a timestamp to signatures and $\mathcal{GPK}$s.

Sign $(\mathcal{GPK}, \mathcal{MSK}, Msg) \rightarrow (Signature)$

1. Select a random integer $r \in \{0,1\}^{l_n/2}$ and $R \in \mathbb{Z}_Q$.
2. Set $u = h^r y_i w_i \bmod n$.

[1] E_i^{-1} is the modular inverse of E in the group of order $p'q'$.

3. Compute

$$\begin{aligned} U_1 &= F^R \bmod P \\ U_2 &= G^{R+x_i} = G^R Y_i \bmod P \\ U_3 &= H^{R+e_i} \bmod P \\ U_4 &= U_1^{s_i} \bmod P \end{aligned}$$

4. Choose

$$\begin{aligned} r_s &\in \{0,1\}^{l_Q+l_c+l_s} \\ r_x &\in \{0,1\}^{l_Q+l_c+l_s} \\ r_r &\in \{0,1\}^{l_n/2+l_c+l_s} \\ r_e &\in \{0,1\}^{l_e+l_c+l_s} \\ R_R &\in \mathbb{Z}_Q \end{aligned}$$

5. Compute

$$\begin{aligned} v &= u^{r_e} f^{-r_s} g^{-r_x} h^{r_r} \bmod n \\ V_1 &= F^{R_R} \bmod P \\ V_2 &= G^{R_R+r_x} \bmod P \\ V_3 &= H^{R_R+r_e} \bmod P \\ V_4 &= U_1^{r_s} \bmod P \end{aligned}$$

6. Compute

$$c = \mathcal{H}(\mathcal{GPK} \parallel u \parallel v \parallel U_1 \parallel U_2 \parallel U_3 \parallel U_4 \parallel V_1 \parallel V_2 \parallel V_3 \parallel V_4 \parallel Msg)$$

7. Compute

$$\begin{aligned} z_s &= r_s + cs_i \\ z_x &= r_x + cx_i \\ z_r &= r_r + c(-r_i - rE_i) \\ z_e &= r_e + ce_i \\ Z_R &= R_R + cR \bmod Q \end{aligned}$$

8. Produce

$$Signature = (c, u, U_1, U_2, U_3, U_4, z_s, z_r, z_x, z_e, Z_R)$$

4.4.4 VERIFY

Basically this phase operates as the namesake one in ACJT. A user, even a non member, may check the validity of a signature using the $\mathcal{GPK}$. The drawback of supporting revocation is that every time a user is going to undertake a verify operation, he must refresh the $\mathcal{GPK}$ with the newest one available. Same advise regarding timestamps, seen in the previous operation, applies here.

Verify $(\mathcal{GPK}, Msg, Signature) \rightarrow (ValidSignature?)$

1. Compute

$$\begin{aligned} v &= (aw)^{-c} f^{-z_s} g^{-z_x} h^{z_r} u^{c2^{l_E}+z_e} \bmod n \\ V_1 &= U_1^{-c} F^{Z_R} \bmod P \\ V_2 &= U_2^{-c} G^{Z_R+z_x} \bmod P \\ V_3 &= U_3^{-c} H^{Z_R+z_e} \bmod P \\ V_4 &= U_4^{-c} U_1^{z_s} \bmod P \end{aligned}$$

2. Check if

$$\begin{aligned} z_e &\in \{0,1\}^{l_e+l_c+l_s} \\ z_s &\in \{0,1\}^{l_Q+l_c+l_s} \\ z_x &\in \{0,1\}^{l_Q+l_c+l_s} \end{aligned}$$

3. Compute

$$c' = \mathcal{H}(\mathcal{GPK} \parallel u \parallel v \parallel U_1 \parallel U_2 \parallel U_3 \parallel U_4 \parallel V_1 \parallel V_2 \parallel V_3 \parallel V_4 \parallel Msg)$$

4. if the check is positive and $c = c'$ the output `true`, otherwise `false`.

4.4.5 OPEN

This operation allows the GM to retrieve the identity of a member from a signed message.

Open $(\mathcal{GPK}, Msg, Signature, \mathcal{GSK}) \rightarrow (m_i)$

1. Check the validity of the group signature with the `VERIFY` procedure.
2. Compute
$$id = U_2 U_1^{-X_G} \bmod P.$$
3. Recover the identity of the member m_i, searching for id in the member list.

4.4.6 REVOKE

This procedure revokes a user membership in the group, so that the member may no longer sign messages. In the first part the GM publishes a secret owned by the revoked member in a MRL and updates the $\mathcal{GPK}$. At a later stage the members update their own secret to continue to be able to prove membership. Thus, we divide this operation in two distinct parts.

Revoke (Group Manager) $(\mathcal{GPK}, m_i) \rightarrow (\mathcal{GPK}, MRL)$

1. GM recovers E_i from the member's data contained in the member list using m_i and publishes it.
2. GM updates the $\mathcal{GPK}$, changing the old w to contain the w_i of the user (where $w_i = w^{E_i^{-1}}$). Now the revoked member m_i cannot prove knowledge of a root of w anymore.

Revoke (Member) $(\mathcal{GPK}, \mathcal{MSK}, MRL) \rightarrow (\mathcal{MSK})$

1. Any member m_j still in the group retrieves from the MRL the published E_i.
2. He then selects α, β so that $\alpha E_i + \beta E_j = 1$.
3. He updates his own $\mathcal{MSK}$ computing $w_j = w_i^{\beta} w_j^{\alpha} \bmod n$.

4.4.7 FULL REVOKE

This procedure allows to revoke anonymity of signatures produced under an old $\mathcal{GPK}$ thanks to the disclosing of s_i. As before, we split the operation into two parts.

Full Revoke (Group Manager) $(m_i) \rightarrow (MRL)$

1. GM recovers s_i from the member's data in the member list using m_i and publishes it in the MRL.

Full Revoke (Member) $(\mathcal{GPK}, Msg, Signature, MRL) \rightarrow (RevokedMemberSignature?)$

1. Check the validity of the group signature with the `VERIFY` procedure.
2. Anyone, user or member, may check if a signature was signed by the revoked member m_i. It is sufficient to check if $U_4^{\frac{P-1}{Q}} = U_1^{\frac{P-1}{Q} s_i} \bmod P$.
3. If the check is positive outputs `true`, otherwise `false`.

Chapter 5

Development of a Group Signature Framework

During our research process we found out that there is no framework that takes advantage of the services offered by group signatures. Thus, the goal of this chapter is to fill the gap left by the cryptographic community and to develop a framework supporting group signatures.

5.1 Development Process

After an initial careful consideration, we decided to adopt a waterfall model for our software development process. Our aim is not to develop a specific application, but, more generally, an experimental framework for group signatures. The requirements of our development, even if very general, are set in stone before the process even starts. We are not even truly interested in a formal verification of the security of our product at the end: our goal is just to show the feasibility of such a framework. Therefore, the waterfall model could probably be the best choice for our situation. In the following, we will describe the following phases:

- Requirements specification
- Design
- Implementation
- Testing
- Deployment

5.2 Requirements Specification

5.2.1 Functional Requirements

- standardized framework for group signatures;
- application independence;

- implementation independence;
- algorithm independence and extensibility;
- the framework should support ACJT scheme;
- the framework should support CG scheme.

Let us briefly comment on about these requirements, since they could appear high-sounding. Once more we should point out our goal: a framework that offers group signature services through standardized interfaces. The idea of a framework is quite obvious: if properly developed, it promotes design and code reuse. It should be application independent too, in the sense that we have no specific client application in mind for its use and we would like to adopt it to a broad range of problems.

Besides, it should be bound neither to any specific group signature scheme nor to a specific scheme implementation. On the contrary, it should be easy enough to add additional schemes or different implementations, without rewriting the framework itself. In other words, we want a framework as low-level as possible, without breaking the abstraction that hides the inner workings of different group signature schemes under it. In addition, to prove the feasibility of our work, we will integrate the two schemes previously analyzed, ACJT and CG.

5.2.2 Non-Functional Requirements

- cross-platform.

Regarding the non-functional requirements, our framework should run ideally on many different platforms, of course on standard desktop computers or notebooks equipped with Windows, MacOS or any flavour of Linux, but since we imagine a future where group signature services will be very important for mobility, it should even be possible to run it on handheld devices, like PDA or cellphones. For this reason, Java, thanks to its high portability, is a good choice as a programming language. We do not set any hardware constraint on the chosen platform as long as it is able to run a Sun's Java Virtual Machine[Mic10b].

5.3 Design

As we can see from the requirements, we are not going to develop a single application but rather a library for group signatures. Since this objective is quite ambitious, at the beginning of our design phase, we researched in the literature and in the public domain for established cryptographic frameworks that could satisfy part or all of our requirements. Basically, we found only two mature choices: Sun's own Java Cryptography Architecture [Mic10a] and the Bouncy Castle Crypto APIs from the Legion of the Bouncy Castle [otBC10]. We decided to analyze the former because it offers an all-ecompassing architecture for supporting cryptography that form the basis even for the latter.

5.3.1 Java Cryptography Architecture

The security of the Java platform is granted by a large set of APIs, implementations and security algorithms and protocols [Mic10c] that provides a standards-based extensible architecture for security and interoperability. The Java Cryptography Architecture (JCA) [Mic10a] is the piece of the architecture that takes care of cryptography and offers APIs for services like digital signatures, message digests, encryption, key management and generation, and secure random number generation, only to name a few. We report from the specifications some key principles that guided the design of JCA, that we find of interest in our case:

implementation independence applications do not need to implement security algorithms. Rather, they can request security services from the Java platform. Security services are implemented in providers, which are plugged into the Java platform via a standard interface;

algorithm independence and extensibility the Java platform includes a number of built-in providers that implement a basic set of security services that are widely used today. However, some applications may rely on emerging standars not yet implemented or on proprietary services. The Java platform supports the installation of custom providers that implement such services.

5.3.1.1 Concepts

The main concepts related to JCA are:

Cryptographic Service a cryptographic primitive, like digital signature, encryption or message digest;

Cryptographic Algorithm a specific algorithm that implements a specific cryptographic service and meets all its requirements, like RSA as digital signature or encryption scheme;

Engine a class that provides the end user with a cryptographic service. It either provides:

- cryptographic operations, like *DigitalSignature*, *Encryption* and *MessageDigest*;
- generators or converters of cryptographic material, like keys and algorithm parameters;
- objects that incorporate cryptographic data to use at higher layers of abstraction, like keystores or certificates.

Example of engine classes are `Signature` or `MessageDigest`, but more or less every concept of modern cryptography is represented, except group signatures of course;

Cryptographic Service Provider (CSP) or more simply Provider. The provider is a package, or a set of packages, that contains the concrete implementations of one or more cryptographic services;

Provider is a class that publishes all the services implemented in the CSP. The use of the same name for both this class and the CSP may cause some confusion in the terminology, but from a pratical point of view the two concepts overlap.

5.3.1.2 Architecture

As we have seen cryptographic services are offered through the definition of engine classes. An engine class defines a high-level Application Programming Interface (API), with public service-generic methods that a client might call, thus achieving algorithm independence. For implementation independence instead, every CSP's implementation class must conform to a fixed interface. Therefore an instance of an engine class should have one or more CSP's implementation classes that have methods with the same signature. Invocation of API methods are routed to the CSP's implementation through a Service Provider Interface (SPI). A SPI is an abstract class, separated from the engine class, that declares an interface that every CSP's implementation of an algorithm must subclass and implement all the abstract methods. The name of a SPI class is the same as the name of the service class suffixed with "Spi": the SPI for the `Signature` engine is `SignatureSpi`.

For example. if we would like to implement a new digital signature algorithm, say "Foo", we must first define the `FooProvider` class that states the link between our implementation and the *Signature* service. Then we must subclass the class `SignatureSpi`, defining our implementation of every abstract methods, for instance a class called `FooSignature`. A client application may then use this implementation through the standard `Signature` engine class.

The focal point of the whole architecture is the `Provider` abstract class. Every engine class contains an instance of this class. Every time we want to add an implementation for a cryptographic service we must subclass this class that contains methods to access the provider's name and version and to register a list of implementations to specific services. In order to be found and usable, a `Provider` must be installed and registered either statically or dinamically. The standard Sun's Java distribution ships already with a good list of providers for the majority of services usually used.

The mechanism for polling and instantiating a CSP's concrete implementation is achieved via the factory and delegation design patterns. In fact, when a particular implementation instance of a specified service is requested by a client, the framework calls the `getInstance()` factory method. The method queries the installed security Providers for that particular implementation of the service and if one Provider's offer matches, an instance of the implementation is then returned to the client. At this point, the engine class performs the task advertised by its interface through delegation to the Provider's implementation instance.

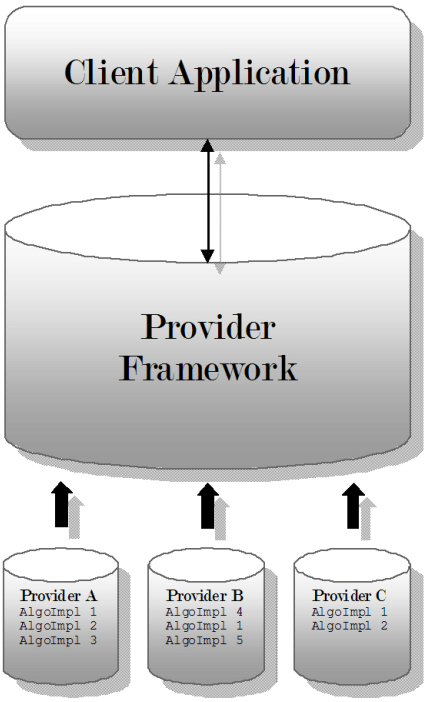

Figure 5.1: JCA Architecture

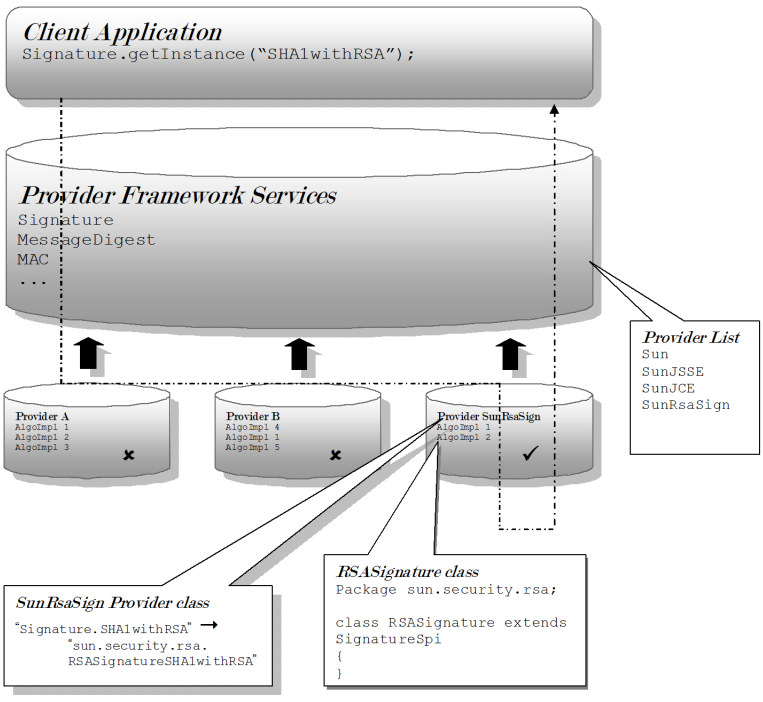

Figure 5.2: JCA Architecture - Signature Example

5.3.1.3 Example: RSA keypair generation

In this first example, we give a pratical example of how is it possible to use the framework to produce a RSA keypair[1][2].

```
// RSA Keypair generation
KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
keyGen.initialize(1024, new SecureRandom());
KeyPair keyPair = keyGen.generateKeyPair();
```

Listing 5.1: RSA Keypair generation

5.3.1.4 Example: RSA digital signature

In this second example we show how is it possible to produce and verify a RSA digital signature, using the RSA keypair generated in previous example.

```
// The message to sign
String message = new String("Lorem ipsum");
byte[] messageBytes = message.getBytes();

// RSA Signature generation
Signature signature = Signature.getInstance("SHA1withRSA");
signature.initSign(keyPair.getPrivate(), new SecureRandom());
signature.update(messageBytes);
byte[] signatureBytes = signature.sign();

// RSA Signature verification
signature.initVerify(keyPair.getPublic());
signature.update(message);
boolean isVerified = signature.verify(signatureBytes);
```

Listing 5.2: RSA Digital Signature generation and verification

5.3.2 Domain Analysis

While the JCA framework gives good insights into how a cryptographic framework should be structured, it is impossible at the present state to map a GS scheme to the services already offered by the JCA. Anyway we would not like to throw away the elegant, even if complex, architecture and if possible we would like to reuse it to give a direction to our work. In fact, thanks to the extendibility of the framework we believe it is possible to define services that we could then integrate into it.

At this point, for the design of the framework, is important to outline:

- high-level data structures to manipulate group signature concepts;
- a group of services that can be traslated to engine classes;
- an interface for every service.

[1] In this particular case we are using one of the default Providers that comes with the standard Java installation. Therefore it is already statically installed and registered.

[2] From now on we will try to keep the code simple. Exception handling will be ignored.

We should thus review the GS schemes in a more abstract way in terms of data structures and operations and extract a small number of services and characterize them in terms of inputs and outputs[3].

We start our revision with the abstraction of data structures required by the GS schemes. We list and describe them here in table 5.1.

Data structure	Description
GSKey	top-level abstraction for a key used in a group signature scheme
GSPublicKey	public key used in a scheme
GSPrivateKey	private key used in a scheme
GSKeyPair	keypair composed by a GSPublicKey and GSPrivateKey
GSGroupCertificate	digital certificate for a $\mathcal{GPK}$
GSMemberCertificate	digital certificate for a $\mathcal{MSK}$
GSSecurityParameters	set of security parameters used in a group
GSIdentity	very high-level abstraction for a group member's identity data
GSRevocationList	Member revocation list

Table 5.1: Group signature data structures

We can now analyze the operations of a GS scheme. We have already noted a dichotomy in our previous review of the schemes. In fact, the schemes are cryptosystems that define a coupling between two different procotols:

- a protocol for "signature-related" operations like sign, verify and open;
- a protocol for group "maintenance" operations, including setup, growing and possibly shrinking of the group.

We use this remark in our favour to start our analysis.

5.3.2.1 Signature-related operations

The first protocol we pinpointed fits perfectly in the set of "cryptographic operations" and is quite similar to the interface of the *Signature* service of the standard library. In our case for `SIGN` and `VERIFY` the inputs are slightly different and we have an additional operation: `OPEN`. However, we can abstract the three operations in the same service since their workflow is similar:

1. initialization with a key (for instance a **GSPrivateKey** or a **GSPublicKey**);
2. loading of the message to be signed/verified/opened;

[3]We note at this point that we will support "out of the box" GS schemes based on SRSA and DDH assumptions (see B.1.1 and B.2.1), since we will factor the basic math directly into the framework. It could be possible however that newer schemes (for example based on bilinear maps) could fit in the framework, even without modifying it, on the condition that they support their own math.

3. execution of the task (passing the signature if verifying or opening);

4. collection of the output.

Following this analysis, the translation from abstact operations to a *GSSignature* service is quite straightforward. We should note that such a service could be used either by a GM or by a group member. We can now follow the sintax seen in listing 5.2 to define in listing 5.3, the methods that compose the interface of our *GSSignature* cryptographic service in terms of the data structures described in table 5.1:

```
// for SIGN
public void initSign(GSPublicKey gpk, GSPrivateKey memberKey,
    GSSecurityParameters params);
public byte [] sign();

// for VERIFY
public void initVerify(GSPublicKey gpk, GSSecurityParameters params
    )
public boolean verify(byte [] signature);

// for OPEN
public void initOpen(GSPublicKey gpk, GSPrivateKey gsk)
public GSIdentity open(byte [] signature);

// accessory method for loading the message
public void update(byte [] messageBytes)
```

Listing 5.3: GSSignature service interface

5.3.2.2 Group maintenance operations

The second protocol is trickier to turn into a service/engine class. The temptation is to manage the "maintenance" of the group outside the framework, however we think the group could fit in guideline: "objects that incorporate cryptographic data to use at higher layers of abstraction", if we stretch the definition. This new service is prerogative of a GM, since he is the only one with the authority and capabilities to create new groups or to enlarge and shrink the set of members. Before defining the interface of the service we still have to solve three issues.

If the previous service was more signature-oriented, this one is more focused on managing members. The first issue is related to data structures. Every scheme solves membership internally with different data structures, so we choose a very high level abstraction to represent a member, `GSIdentity`. This data structure contains the identity of the member and some kind of data representing membership proof. Since outside the framework this identity has very little meaning, we will manage the list of members inside the framework. The opposite choice is even possible to gain flexibility but losing algorithm independence. For the same reason, we will handle the member revocation list, $\mathcal{MRL}$, inside the framework.

For security reasons we believe the GM should always be kept offline in a network: he takes care of the $\mathcal{GSK}$ and the list of member identities: both of them should be kept secret. Furthermore, except for the `JOIN` action he has not direct contact with users, members or non-members alike. In fact, the users are

only interested in up-to-date $\mathcal{GPK}$ and $\mathcal{MRL}$, which can be published anywhere without the need for the GM to be online. The `JOIN` operation arises the second issue since it is conceptually different: it is a n-round protocol between the GM and a would-be-member. In our opinion, this exchange should be executed on a secure and private channel and entirely offline. The model we propose is similar to the subscription of a new mobile contract: the user goes physically to an authorized vendor, signs a contract and interactively gets some kind of proof of membership in return. We are aware that there are many problems connected to this choice that can even break the security of the schemes, but we think it to be safer even if not less problematic than the online scenario. We will not dwell on this matter and we will consider the `JOIN` procedure as part of our service but without the requirement of being a protocol between different parties. A proper solution to this problem will not be part of this work but will be considered as an future extension.

The last issue concerns revocation. The sharp reader might already have noticed that both `REVOKE` operations are only partially responsibility of the GM. For the simple `REVOKE` every member still in good standing for membership has to make a bit of computation to prove that he has not been removed from the group. We will move this share of the operation in a new service, only for members. On the contrary, `FULL-REVOKE` for a member could be considered as a special version of `OPEN`, so we will add it in the `GSSignature` service.

After solving our last concerns we are now ready to define two additional services and their interfaces to our framework: *GSGroupManager* and *GSMemberManager* in listings 5.4 and 5.5 respectively. The extension of the interface of *GSSignature* could be found in listing 5.6.

```
// for SETUP
public GSKeyPair doSetup(GSSecurityParameters params);
public GSGroupCertificate buildGroupCertificate(GSPublicKey gpk,
    String issuer, String group, Date startDate, Date expiryDate,
    int serialNumber, GSSecurityParameters params);

// for JOIN
public GSMemberCertificate joinGroup(String member,
    GSGroupCertificate groupCert, GSPrivateKey gsk)

// for REVOKE
public GSPublicKey doRevoke(GSPublicKey gpk, GSIdentity member);

// for FULL-REVOKE
public void doFullRevoke(GSIdentity member);

public GSRevocationList getRevocationList();
```

Listing 5.4: GSGroupManager service interface

```
// for REVOKE
public GSPrivateKey updateMemberKey(GSPublicKey gpk,
    GSRevocationList list, GSPrivateKey gsk)
```

Listing 5.5: GSMemberManager service interface

```
// for FULL-REVOKE
public void initRevoke(GSPublicKey gpk, GSRevocationList list)
public boolean revoke(byte [] signature);
```

Listing 5.6: GSSignature service interface extension

5.4 Implementation

In this section we will outline the implementation phase of our software development process. There will be two main parts: the first related to the framework itself and the second to the implementation of the ACJT scheme into the framework.

5.4.1 Framework

5.4.1.1 Data Structures

We implemented the framework following a bottom-up approach starting first with the data structures. In fact, we traslated into Java interfaces, abstract classes or final classes the data structures specified during the design phase. They should provide little or no code at all and be very abstract, since they represent concepts that every scheme should override or redefine. In addition, the structures related to keys mimic the case of the corresponding interfaces of the standard JCA framework, like `Key`. This is the case of `GSKey`, that should be only considered as a way to group cryptographic keys and to provide type safety. We report the implementation of some data structures as example in listings 5.7, 5.8, 5.9, 5.10 and 5.11.

```
import java.security.Key;

public interface GSKey extends Key
{
        public int getSize();
        public String getAlgorithm();
}
```

Listing 5.7: GSKey interface

```
import java.security.PrivateKey;

public interface GSPrivateKey extends GSKey, PrivateKey
{

}
```

Listing 5.8: GSPrivateKey interface

```
import java.security.PublicKey;

public interface GSPublicKey extends GSKey, PublicKey
{

}
```

Listing 5.9: GSPublicKey interface

```
import java.security.KeyPair;

public final class GSKeyPair
{
        private GSPublicKey publicKey;
        private GSPrivateKey privateKey;

        public GSKeyPair(GSPublicKey publicKey, GSPrivateKey
            privateKey)
        {
                this.publicKey = publicKey;
                this.privateKey = privateKey;
        }

        public GSKeyPair(KeyPair keyPair)
        {
                this.publicKey = (GSPublicKey) keyPair.getPublic();
                this.privateKey = (GSPrivateKey) keyPair.getPrivate
                    ();
        }

        public GSPublicKey getPublic()
        {
                return publicKey;
        }

        public GSPrivateKey getPrivate()
        {
                return privateKey;
        }
}
```

Listing 5.10: GSKeyPair final class

```
import java.security.spec.AlgorithmParameterSpec;

public interface GSSecurityParameters extends
    AlgorithmParameterSpec
{
        public byte [] getEncoded();
        public String toString();
}
```

Listing 5.11: GSSecurityParameters interface

As we have seen the implementation of these structures is very straightforward. The case of certificates is more complex: they should group keys with additional informations, such as validity time and scope, and optionally, be signed with long-term credentials (like standard digital signatures). Once again we prefer to imitate already established cases instead of putting forward new formats. For this end, we chose a X.509-like format for the representation of our certificates. Following X.509 certificate structure, `GSGroupCertificate` is a wrapper around:

- Version
- Serial Number
- Group Signature Algorithm

- Issuer
- Validity
  - Not Before
  - Not After
- Subject
- Timestamp
- `GSPublicKey`
- `GSSecurityParameters`
- Digital (long-term) Signature

The structure `GSMemberCertificate` is quite similar, but with a main difference: it can not be signed since the $\mathcal{MSK}$ is prone to change during its life due to changes made by members. Thus, signing makes no sense. With this premise we see here the structure of a `GSMemberCertificate`:

- Version
- Serial Number
- Group Signature Algorithm
- Issuer
- Validity
  - Not Before
  - Not After
- Subject
- Timestamp
- `GSSecurityParameters`
- `GSPrivateKey`

The last structure to be implemented, `GSIdentity`, is probably the most abstract. Basically, it is just a "blob" of membership data wrapped around the name of the member, because we do not want to expose unnecessary details related to a specific scheme. We even said that we will manage identities "inside" the framework. The idea is simple: we will save every member identity in a list structure and allow every scheme to provide efficient and specific search algorithms for the list. We see the code in listing 5.12.

```
public class GSIdentity
{
        protected String subject;
        protected Object [] data;

        public GSIdentity(String subject, Object [] data)
        {
```

```
                this.subject = subject;
        }

        public String getName()
        {
                return subject;
        }

        public Object [] getData()
        {
                return data;
        }

        public void setData(Object [] data)
        {
                this.data = data;
        }
}
```

Listing 5.12: GSIdentity class

5.4.1.2 Engine Classes

Having described the data structures involved in our framework implementation, we can now focus our attention on the implementation of the engine classes. As we already noted during the design phase, algorithm and implementation independence are achieved by defining types of cryptographic services and the related engine classes that provide their functionalities. This job is actually split in two parts. The first is directed client-side with the implementation of the outer interface, the second to the definition of the related SPI class the every provider supplying that service must implement.

The complex part is however the integration of the two. The JCA uses two cooperating design patterns for the collaboration of the parts: *abstract factory* and *delegation*.

Abstract Factory provides a way to encapsulate a group of individual factories. The client software creates a concrete implementation of the abstract factory and then uses the generic interface to create the concrete object. The client may request a particular flavour of the concrete object being created but in any case he uses only the generic interfaces of the object.

Delegation provides a mean for an object (the delegator) to delegate a task to an associated helper object (the delegate). This pattern is not only important by itself but even as the building block behind composition.

To show the actual implementation we provide here only the class diagrams of the two sample services, *GSSignature* in figure 5.3 and *GSGroupManager* in figure 5.4. We advise the reader to refer to the code for more details.

5.4.2 Group Signature Scheme

In this section we will analyze the implementation of ACJT, as an example of how a group signature scheme may be integrated in our framework.

The actions required are resumed here:

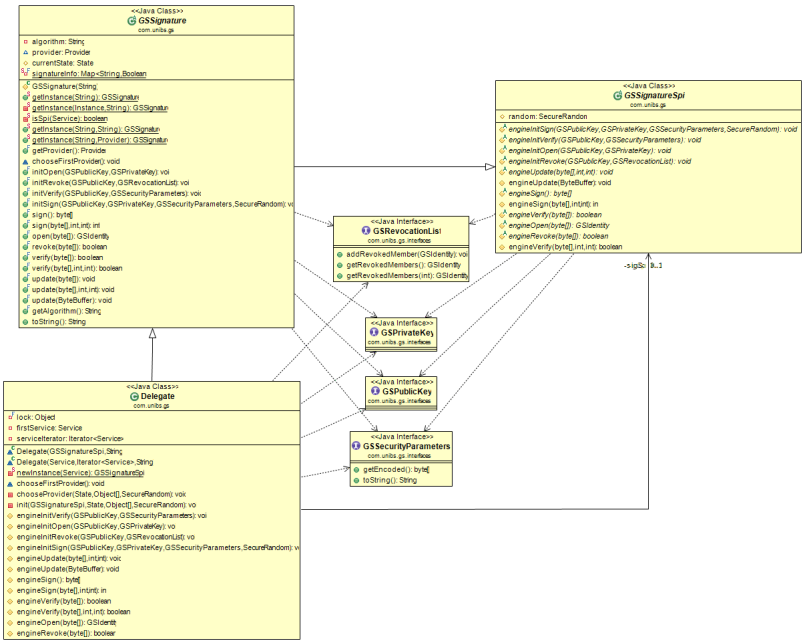

Figure 5.3: GSSignature service class diagram

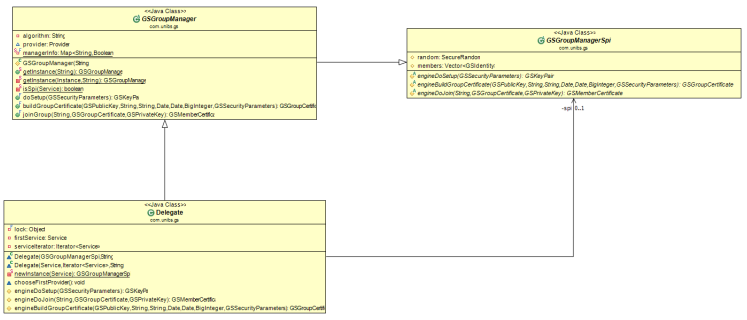

Figure 5.4: GSGroupManager service class diagram

- definition of the Provider;
- implementation of the data structures;
- implementation of the service(s).

5.4.2.1 Provider definition

The `Provider` class is the binding agent between the framework and the algorithm implementation. It defines which services offer that particular implementation. We illustrate in listing 5.13 the class definition.

```
import java.security.Provider;

public class ACJTProvider extends Provider
{
        private static final long serialVersionUID =
            378060836875303917 4L;
        private static final String NAME = "ACJT";
        private static final double VERSION = 2.0;
        private static final String INFO = "ACJT Group Signature
            Provider by Diego Ferri 2010";

        public ACJTProvider()
        {
                super(ACJTProvider.NAME, ACJTProvider.VERSION,
                    ACJTProvider.INFO);
                put("GSSignature.ACJT", "com.unibs.gs.acjt.
                    ACJTSignature");
                put("GSGroupManager.ACJT", "com.unibs.gs.acjt.
                    ACJTGroupManager");
                put("KeyPairGenerator.ACJT", "com.unibs.gs.acjt.
                    ACJTGroupKeyPairGenerator");
        }

}
```

Listing 5.13: ACJTProvider class

From the code we gather that three services are registered: the framework's `GSSignature` and `GSGroupManager` and in addition a `KeyPairGenerator`. The latter is a service from the standard package for which we provide an algorithm implementation. The target of the service is to produce a group public/private keypair given the securoty parameters. In our case it matches the `SETUP` operation of the scheme. Of course, we do not register any `GSMemberManager` service, since in ACJT there is no need to update the $\mathcal{MSK}$. This is a plus of this architecture: during runtime this situation will be dealt and a standard exception will be thrown to indicate that no actual implementation of the service exists or is registered.

5.4.2.2 Data Structure and Service Implementation

It is mandatory to redefine the abstract, generalistic framework's data structures. However, even in this case our data structures are very simple. Their main contribute to the project is to convert cryptographic keys, certificates and parameters from bytes to structures and viceversa. The structures implemented are listed in table 5.2.

Data structure	Description
ACJTKey	abstraction for a key to provide type-safety
ACJTPublicKey	public key interface to provide type-safety
ACJTPrivateKey	private key interface to provide type-safety
ACJTGroupPublicKeyImpl	group public key
ACJTGroupPrivateKeyImpl	group private key
ACJTMemberKeyImpl	member secret key
ACJTGroupCertificate	group certificate
ACJTMemberCertificate	member certificate
ACJTSecurityParameters	security parameters
ACJTIdentity	group member's identity data

Table 5.2: ACJT data structures

As for the framework code we only outline the service implementations with the class diagrams, see figures 5.5 and 5.6. We advise the reader to refer to the code for more details.

5.4.3 Package Organization

In table 5.3, 5.4 we list the package structure of the framework and of ACJT implementation. Even in this case, when possible, we mimic JCA's structure.

Package	Content

com.unibs.gs	`GSSignature`
	`GSSignatureSpi`
	`GSGroupManager`
	`GSGroupManagerSpi`
	`GSMathCore`

com.unibs.gs.interfaces	`GSKey`
	`GSKeyPair`
	`GSPublicKey`
	`GSPrivateKey`
	`GSRevocationList`
	`GSSecurityParameters`
	`GSIdentity`

Table 5.3: Framework package organization

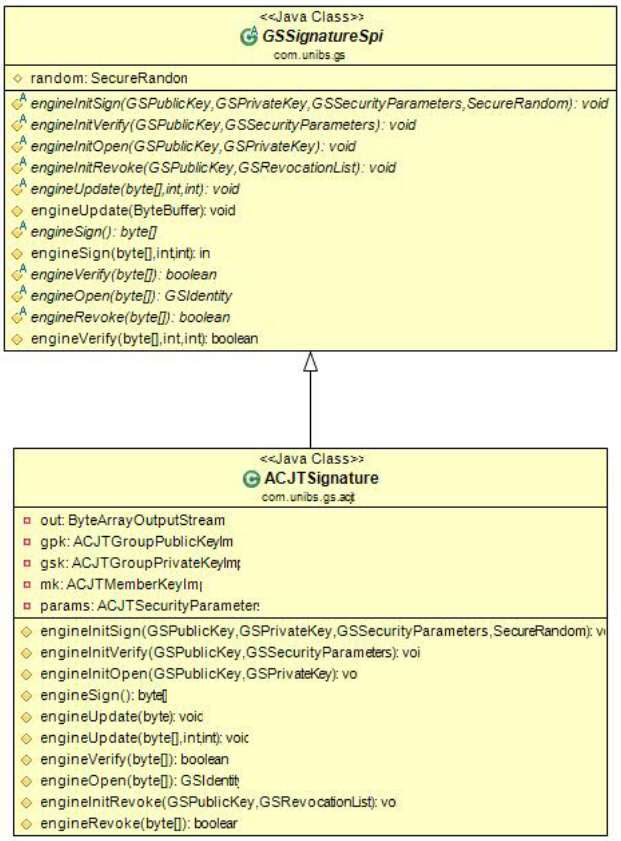

Figure 5.5: ACJTSignature service class diagram

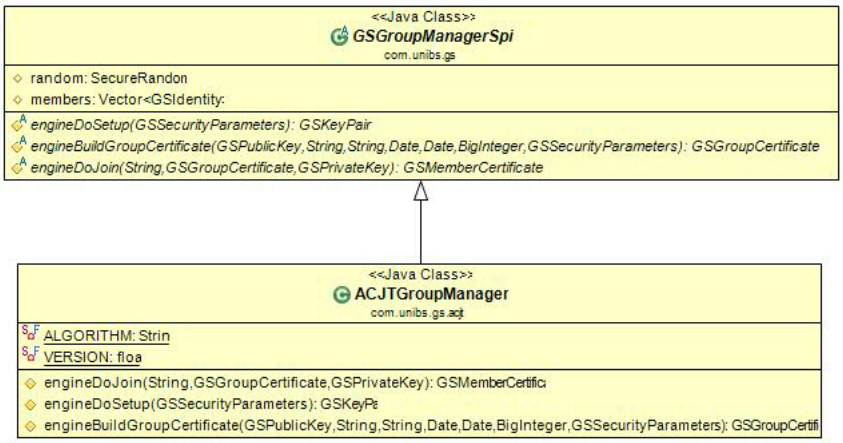

Figure 5.6: ACJTGroupManager service class diagram

Package	Content
com.unibs.gs.acjt	ACJTGroupKeyPairGenerator ACJTGroupManager ACJTGroupPrivateKeyImpl ACJTGroupPublicKeyImpl ACJTMemberKeyImpl ACJTProvider ACJTSecurityParameters ACJTSignature ACJTSignatureImpl
com.unibs.gs.acjt.certs	ACJTGroupCertificate ACJTMemberCertificate
com.unibs.gs.acjt.interfaces	ACJTKey ACJTPrivateKey ACJTPrivateKey

Table 5.4: ACJT package organization

5.5 Testing

Testing was performed using a black box approach. Our aim was to test the functionality of the whole system, from an external point of view. Since the structure of engine classes, SPI classes and providers, this phase could be considered integration and system testing at the same time.

For determining the inputs and outputs of the classes, we employed techniques such as *equivalence partitioning* and *boundary-value analysis*.

The testing activity revealed a tight coupling between tests. Many methods need to be performed in a sequential manner, this behaviour mirrors the sequence of usual operations needed for the use of group signatures. For instance, for testing *Open* it is mandatory to test this sequence: `SETUP` then `JOIN` then `SIGN` then `OPEN`.

All tests were successful and performed using JUnit Testing Framework.

5.6 Deployment

The framework and both group signature schemes ship together as a *JAR* (JAva ARchive) file. Usual directions about using a *.jar* file apply here. We tested the package with a Java Runtime Environment (JRE) version 6 Update 19.

5.7 Tools

The use of Java as programming language, as dictated by the requirements, led to the choice of using *Eclipse* as integrated development environment. For testing, we used the *JUnit* testing framework as a fully integrated solution in

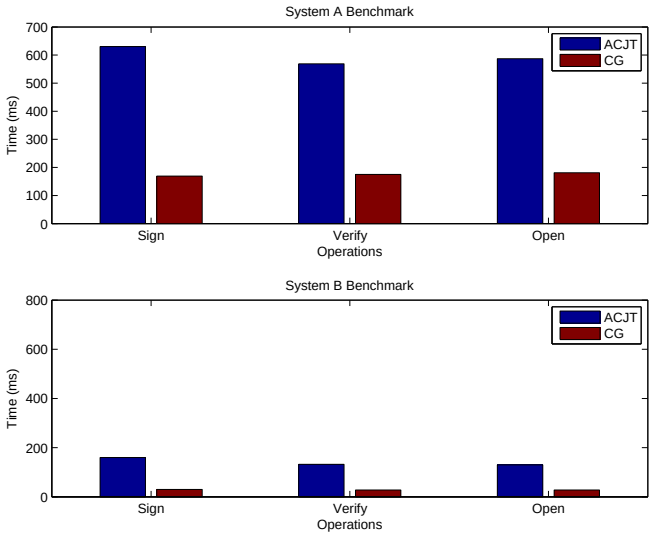

Figure 5.7: Group signature framework implementation benchmarks

Eclipse. As compiler and interpreter of the Java language the obvious choice is the Sun's *Java Development Kit*. We summarize the tools in the table 5.5.

Tool	Description	Version
Eclipse	IDE	3.5.1
JUnit	Testing Framework	4.5.0
JDK	Sun's Java Development Kit	6 Update 19
Subversion	Advanced control version system	1.6.6
TortoiseSVN	Windows client for Subversion	1.6.6

Table 5.5: Development tools

5.8 Benchmarks

We report in graph 5.7 a brief benchmark of our implementation to give an idea of the time required to the final user for common group signature operations. The values are an average of 5 test runs, excluding the first just after the JVM is started. As security parameters we use a 1024 bits comparable security. All values are expressed in milliseconds.

For our tests we used two systems:

(A) notebook equipped with single-core Intel Pentium M760 processor at 2.0 GHz with 2 Mb of L2 cache and 2 Gb of DDR PC-2700 ram;

(B) notebook equipped with quad-core Intel Core i7 720QM processor at 1.60 GHz with 6 Mb of L3 cache and 4 Gb of DDR3-1333 ram.

5.9 Conclusions

For the sake of brevity and because we would like to focus more on the network security aspects, we only outlined the development process. We left out of the chapter the use case description, the package diagrams, the sequence diagrams and the list of test cases. In addition we omitted the design of our own math library, included in the project.

In any case, we described in this chapter the phases of our software development process. In addition, even if we tagged our framework as experimental, we smoothly implemented both ACJT and CG in it. The next step will be the integration in the PERIMETER (see chapter 7) project: an activity already planned in the near future.

The source code of our library can be found on the CD-ROM support included in our work.

Chapter 6

Performance Analysis and Evaluation

In this chapter we will analyze the scalability constraints of the presented group signature schemes. We will split this analysis in two parts. In the first we consider the burden imposed on a single peer, then we will take into account the bandwidth occupation on the whole network. Together with ACJT, we choose the full revocation version of CG, as the object of our analisys.

6.1 Single Peer Model

In this section we define a simple model to analyze performances of the group signature main operations. In this part we analyze the resources of a single node: time and space occupation. We are marginally interested in the time complexity of the operations, mostly to check if they are independent of the number of users. We are much more interested in the space occupation of their products, because this will form the basis for our system-wide analysis.

6.1.1 Time Complexity

6.1.1.1 SETUP and JOIN

For security reasons we believe the GM should be always kept offline in a network: it takes care of the $\mathcal{GSK}$ and the list of member identities: both of them should be kept secret.

The `SETUP` algorithm is run by the GM alone, only at the creation of the group. Here the main parameters involved are the security parameters of the group, as described by the group signature scheme itself. Both schemes offer different security parameters. The only common point is the size of the RSA modulus used: the other parameters are bounded by linear relations to the size of the modulus. Thus the computation time required by the `SETUP` algorithm is only influenced by the size in bit of the RSA modulus, in an exponential way.

The `JOIN` algorithm is a n-round protocol between the GM and a would-be-member. In our opinion, this exchange should be executed on a secure and private channel and entirely offline: the model we propose is similar to the

subscription of a new mobile contract. In this case the user goes physically to an authorized vendor, signs a contract and gets some kind of proof of membership in return. We are aware that there are many problems connected to this choice that even breaks the security of the schemes, but we think it to be safer and not less problematic than the online scenario. Even in this case, however, the computation time is only bound to the size of the RSA modulus, without any constraint on the number of members.

For the presented reasons both `SETUP` and `JOIN`, from now on will be considered offline operations.

6.1.1.2 SIGN

Both ACJT and CG schemes feature a `SIGN` procedure where the computational effort required to sign is independent with respect to the number of members of the group, that is of constant time, $O(1)$.

6.1.1.3 VERIFY

Both ACJT and CG schemes feature a `VERIFY` procedure where the computational effort required to verify is independent with respect to the number of members of the group, that is of constant time, $O(1)$.

6.1.1.4 OPEN

Both ACJT and CG schemes feature a `OPEN` procedure where the computational effort required is independent with respect to the number of members of the group, that is of constant time, $O(1)$.

6.1.1.5 REVOKE

Only CG scheme provides revocation capability. In this operation the GM must update the $\mathcal{GPK}$ so that a revoked member may no longer prove membership in a signature. The change involved in one revocation is of constant size in terms of number of members in the group. The members of the group must then update their $\mathcal{MSK}$ too, with a change of constant size. The latter operation can be delayed till signing is required, but in that case it is linear in the number of revoked members since last update of the $\mathcal{MSK}$.

6.1.1.6 FULL REVOKE

Only CG scheme provides full revocation capability. In this operation the GM publishes a token of a revoked member (in a member revocation list) so that any other user may check if a signature has been produced by that member. Both operations are of constant size with respect to every parameter.

6.1.2 Space Occupation

6.1.2.1 SETUP and JOIN

Both phases are meant to be run offline. In any case, as for the time complexity, the size of the keys and certificates produced is related only to the size of the RSA modulus, once again in an exponential way.

6.1.2.2 SIGN

Both ACJT and CG schemes feature a `SIGN` procedure where the size of the signature attached to a message is independent of the message size and of the number of members of the group, that is of constant size, $O(1)$. The size of the signature, thus the security of it, depends linearly on the bit-size of the RSA modulus used.

6.1.2.3 VERIFY

In both scheme, the `VERIFY` procedure is of constant size with respect to the number of members of the group.

6.1.2.4 OPEN

In both scheme, the `OPEN` procedure is of constant size with respect to the number of members of the group.

6.1.2.5 REVOKE

Only CG scheme provides revocation capability. During this operation the GM publishes one number half the size of the RSA modulus maximum; in addition, every member still in good standing must look up this number from the member revocation list and update his own $\mathcal{MSK}$.

6.1.2.6 FULL REVOKE

Same as `REVOKE` operation, but without the update of the $\mathcal{MSK}$.

6.2 Distributed System Model

We analyze now both schemes at a broader, system level taking into account the single peer examination of the previous section. At this level, the only scarce resource is bandwidth. We define the bandwith $\mathfrak{B}$ as the total amount of traffic moved on the network in a time unit. Our definition of time unit is not strict but we imagine it as small as one second. We will evaluate how much bandwidth is needed by the schemes in terms of number of messages signed, of verifiers and total users, in a multiple-verifiers environment. Our attention focuses on the set of online members in the group, that is the set of users that are signing or verifying in a time unit. In both examinations we skip the `SETUP` and `JOIN` operations because, being executed offline, they have no influence over bandwidth. Later we will cull even the `OPEN` procedure from the analysis. The other operations, as we have riaffirmed in previous section, are independent (or linear) of the total number of users in the group, but are exponential in the size of RSA modulus chosen. We see the burden that every operation has on total bandwidth utilization.

6.2.1 ACJT Model

We report here the variables needed to model the ACJT scheme bandwidth utlization:

L_n
: RSA modulus size

L_m
: average message length

L_s
: average signature length (depends on L_n)

L_{GPK}
: average GPK length (depends on L_n)

n_s
: average number of signed messages produced by every user in the time unit

n_v
: average number of verifiers for every signed message in the time unit

p_a
: percent of active users in a given moment

p_o
: percent of opened signatures in the time unit

N
: number of users in the group at the beginning of the time unit

α
: average number of users joining the group in the time unit

N_a
: number of online/active users in the group at the end of the time unit, where $N_a = (N + \alpha) * p_a$

N_m
: total number of messages circulating on the network in a time unit, where $N_m = N_a * n_s * n_v$

6.2.1.1 SIGN

This factor includes the total bandwidth consumed by signed messages spread over the network. It is linear on the total number of messages in the network.

$$\Sigma = N_m * (L_m + L_s)$$

6.2.1.2 VERIFY

To verify the validity of an alleged signature a user needs the $\mathcal{GPK}$. This procedure can greatly benefit from a caching mechanism, assuming that the verifier population does not vary too much over time, because the $\mathcal{GPK}$ never changes. In a real world scenario this term could be nearly of costant size. As always, in case of indecision we choose a conservative approach: in this model we do not consider the impact of such an optimization.

$$\Phi = L_{GPK} * N_m$$

6.2.1.3 OPEN

The cost is:

$$O = L_{GPK} * p_o * N_m * n_s$$

As we expect that the number of malicious signed messages in the network will always be a fixed percentage of the total number of messages, we decide to express p_o as a percentage of N_m. This cost is valid only in abstract, because the bandwidth incurs in no additional costs if the membership manager and the revocation manager are the same, as the default setting for the scheme.

6.2.1.4 ACJT Bandwidth Utilization

Bandwidth needed:

$$\mathfrak{B} = \Sigma + \Phi + O$$

6.2.2 CG Model

The CG scheme features the same operations of the ACJT scheme and extends it with revocation. It adds a new set of parameters and shares the ones already described. We report here only the new or modified ones.

L_r
: average revocation token length (depends on L_n)

ρ
: average number of revoked users in the time unit

N_a
: number of online/active users in the group at the end of the time unit, where $N_a = (N + \alpha - \rho) * p_a$

6.2.2.1 SIGN

$$\Sigma = N_m * (L_m + L_s)$$

6.2.2.2 VERIFY

As we have discussed in the previous section caching could be a great bonus for the `VERIFY` operation in ACJT. In the CG scheme instead the benefit is nearly nullified due the periodical modification of the $\mathcal{GPK}$ caused by revocations. In this case, in fact, every user (member or non-member) who wants to verify a signature must always check if the $\mathcal{GPK}$ has changed in the meantime.

$$\Phi = L_{GPK} * N_m$$

6.2.2.3 OPEN

The cost is:

$$O = L_{GPK} * p_o * N_m * n_s$$

As we expect that the number of malicious signatures in the group will always be a fixed percentage of the total number of messages, we decide to express p_o as a percentage of N_m. This cost is valid only in abstract, because the bandwidth

incurs in no additional costs if the membership manager and the revocation manager are the same, as the default setting for the scheme.

6.2.2.4 REVOKE

In a time unit, every active member of the group must download from a $\mathcal{MRL}$ a token of size L_r for every revoked member to update his witness. This update can be delayed till a signing is required but, obviously, in that case, will be linear in the number of revoked members since last update.

$$K = N_a * L_r * \rho$$

6.2.2.5 FULL REVOKE

Same as `REVOKE`.

$$K = N_a * L_r * \rho$$

6.2.2.6 CG Bandwidth Utilization

Bandwidth needed:

$$\mathfrak{B} = \Sigma + \Phi + O + K$$

6.3 Evaluation

The goal of this section will be to evaluate the model performance under various conditions: we will try to collapse the system to eventually find constraints that can limit its scalability.

Before evaluating our model, we still need to define values for the security parameters of the two different signature schemes and add some others for the consistency of the environment: one set of parameters related to the properties of a communication protocol and another one to the user population.

We choose a moderate security level based on a 1024 bits RSA modulus and set the security parameters of both schemes to a comparable level of security[1]. We derive the average lengths of signatures, L_s, and $\mathcal{GPK}$, L_{GPK}, from the analysis of the schemes (see tables 6.1 and 6.2).

After that we indicate the values used for the security parameters of both schemes in tables 6.3 and 6.4.

In relation to the communication protocol, we still have to pinpoint the parameters regarding message size and the ratio between sign, verify, open and revoke actions. For this reason we define a believable model protocol where a member should produce and sign n_s messages of length L_m bits in a time unit, each of them verified by n_v different parties. Our approach is fairly conservative since we consider every active user in the network to actively sign and verify messages every time unit. Finally we fix a constant percentage, p_o, of malicious messages.

[1] We should point out that the modulus in ACJT should be greater than CG's. In fact in ACJT, the GM knows the value $a^{x_i}a_o \bmod n$ by a member, with x_i being the member's secret. But since he knows the factorization of n, he has an advantage computing discrete logarithms modulo n. Hence, to protect x_i ACJT's modulus should be greater than CG's.

ACJT	
Parameter	Length
n, a, a_0, y, g, h	$12l_p$
$\mathcal{GPK}(L_{GPK})$	$12l_p$
c	k
T_1, T_2, T_3	$6l_p$
s_1	$\epsilon(\gamma_2 + k) + 1$
s_2	$\epsilon(\lambda_2 + k) + 1$
s_3	$\epsilon(\lambda_1 + 2l_p + k + 1) + 1$
s_4	$\epsilon(2l_p + k) + 1$
$Signature(L_s)$	$6l_p + k + \epsilon(4l_p + 4k + \lambda_1 + \lambda_2 + \gamma_2 + 1) + 4$

Table 6.1: Analytical analysis of the lengths of signatures and $\mathcal{GPK}$ of ACJT scheme

As our aim is to test the scalability and feasibility of the proposed schemes in a largely distributed system with respect to the features of a model protocol and to the user population, we will test our model in six different scenarios varying

A the security level to analyze the overhead introduced by signatures;

B the number of signed messages produced in a time unit;

C the number of verifiers for every signed message;

D the number of total members partecipating in the group;

E the number of non-members added to the group;

F the number of members leaving the group because of revocation.

At this time we can cull even the `OPEN` cost on bandwidth because in our implementation the group manager and the revocation manager coincide.

6.3.1 Scenario A: Signature overhead

We define the overhead introduced by a group signature in a message as the percentage

$$\omega = \frac{L_s}{L_m + L_s} \cdot 100$$

We show in the figure 6.1 the overhead introduced in a signed message as the size of message changes, with four different levels of security. The length of the signatures is calculated using tables 6.1 and 6.2.

6.3.2 Scenario B: Number of signed messages

In this test we vary the number of signed messages produced by each active user in a time unit. We see in table 6.5 the values and the interval ranges adopted in the test.

CG[a]	
Parameter	Length
n, a, g, h, w, f	$6l_n$
Q	l_Q
P, F, G, H	$4l_P$
$\mathcal{GPK}(L_{GPK})$	$6l_n + 4l_P + l_Q$
c	l_c
u	l_n
U_1, U_2, U_3, U_4	$4l_P$
z_s	$l_Q + l_c + l_s$
z_x	$l_Q + l_c + l_s$
z_r	$l_n + l_c$
z_e	$l_e + l_c + l_s$
Z_R	l_Q
$Signature(L_s)$	$2l_n + 4l_P + 3l_Q + 5l_c + 3l_s + l_e$

[a] Full revocation version

Table 6.2: Analytical analysis of the lengths of signatures and $\mathcal{GPK}$ of CG scheme

6.3.3 Scenario C: Number of verifiers for every signed message

In this test we vary the number of verifiers for every signed message produced by each active user in a time unit. We see in table 6.6 the values and the interval ranges adopted in the test.

Understandably the results of mirroring the number of signed messages and of verifiers produce the same results because in both cases the total number of messages in the network remains the same. We report in figures 6.2 and 6.3 the results of applying to the model the proposed values.

6.3.4 Scenario D: Number of total members

In this test we vary the total number of members of the group. This choice has influence over the number of active users and in cascade over the total number of messages in the network. We could achieve the same result modifying the percentage of active population of a group, p_a. We see in table 6.7 the values and interval ranges adopted in the test with a chancing number of total members.

We report in figures 6.4 and 6.5 the results of applying to the model the proposed values.

6.3.5 Scenario E: Number of non-members joining to the group

In this test we measure the effect on bandwidth of adding a set of new member to the group. We see in table 6.8 the values and interval ranges adopted in the

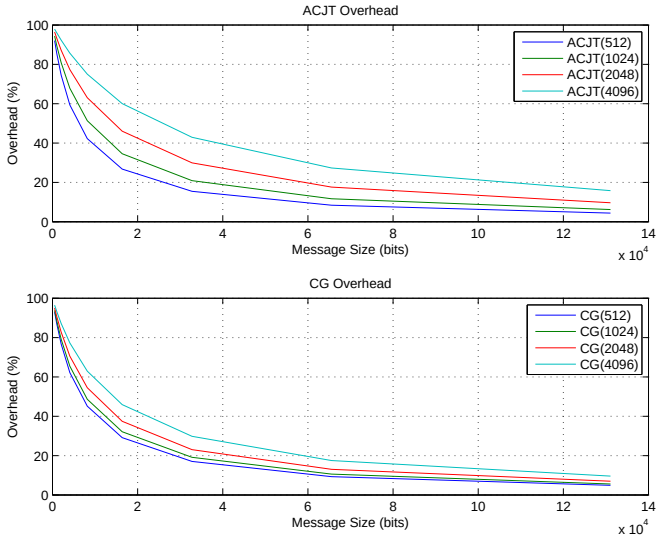

Figure 6.1: Scenario A: Overhead introduced by signatures as the size of message varies

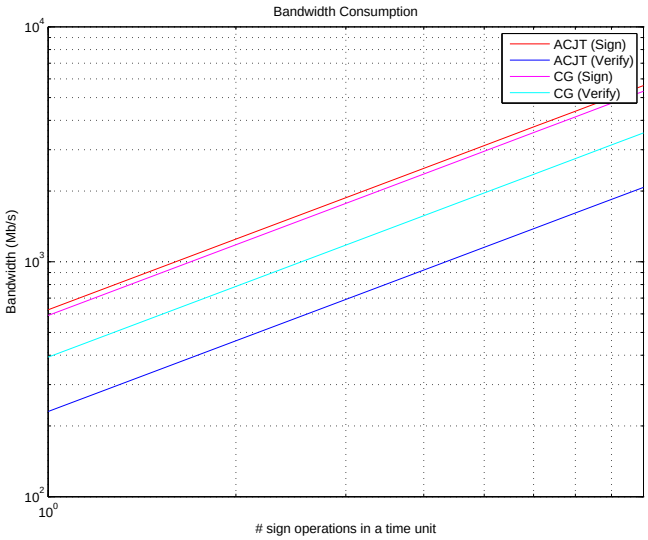

Figure 6.2: Scenario B: Bandwidth cost as the number of signed messages in a time unit changes

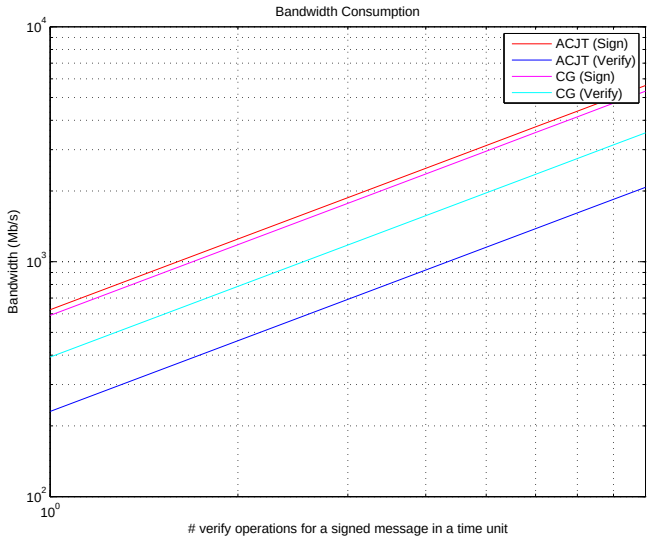

Figure 6.3: Scenario C: Bandwidth cost as the number of verfiers for every signed message in a time unit changes

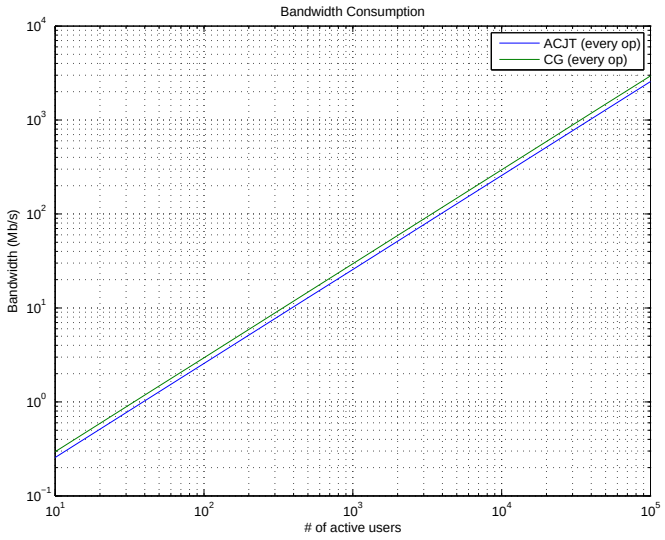

Figure 6.4: Scenario D: Bandwidth cost as the total member number changes

ACJT	
Parameter	Value
l_p	512
ϵ	1.1
λ_1	838
λ_2	600
γ_1	1102
γ_2	800
k	160
$\mathcal{H}$	SHA-1
$\mathcal{GPK}(L_{GPK})$	6144
$Signature(L_s)$	8655

Table 6.3: ACJT security parameters adopted for model evaluation

test with a variable number of new members joining the group.

We report in figures 6.6 and 6.7 the results of applying to the model the proposed values.

6.3.6 Scenario F: Number of members leaving the group

We see in table 6.9 the values and interval ranges adopted in the last test with a variable number of members being revoked. This test is obviously run only for CG scheme.

We report in figures 6.8, 6.9 and 6.10 the results of applying to the model the proposed values. In the last figure we can note the linear dependence of the bandwith with respect to the number of revoked member since last update.

6.4 Conlusions

As we can see the schemes behave quite similarly for standard signing or verifying: both exhibit an asymptotical linear progression in terms of bandwidth utilization.

Bearing in mind this outcome, the choice of one scheme over the other is quite difficult. In fact, as we can see in 6.1, until the security requirements are low, ACJT introduces a lower overhead than CG. Roughly speaking, if the group signature modulus is equal or less than 1024 bits ACJT performs better than CG, especially if the verifier/signer ratio is high. If the modulus is 2048 bits ACJT could still be preferred if a proper caching mechanism is in place and if the group of verifiers is quite stable. Above this threshold the results are uncertain: CG features a smaller size of signatures but ACJT could still be equal or better thanks to caching, if the number of verifiers for signed message is high and the group of verifiers does not change much. From a network utilization point of view ACJT is probably the winner of the contest.

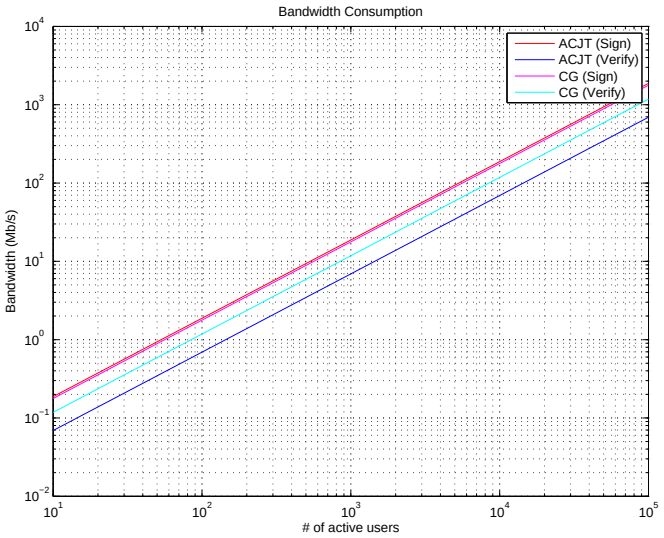

Figure 6.5: Scenario D: Bandwidth cost as the total member number changes (details of every operation)

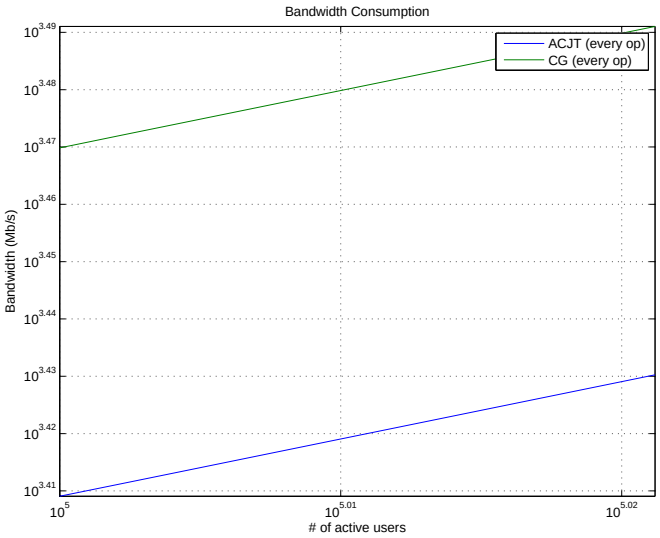

Figure 6.6: Scenario E: Bandwidth cost as the joined member number changes

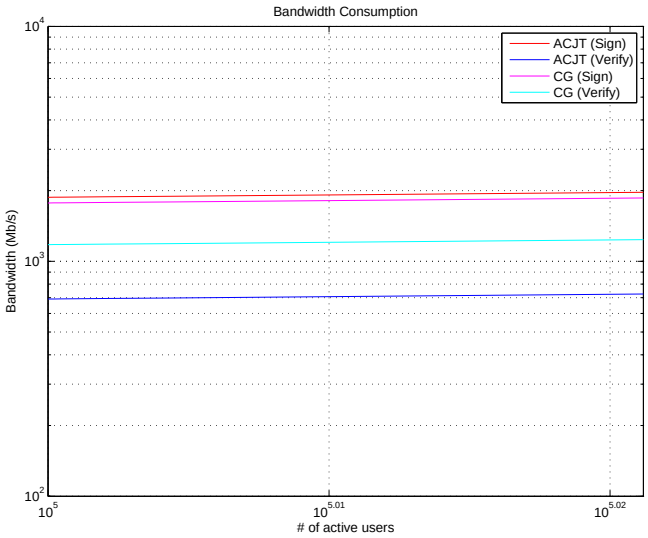

Figure 6.7: Scenario E: Bandwidth cost as the joined member number changes (details of every operation)

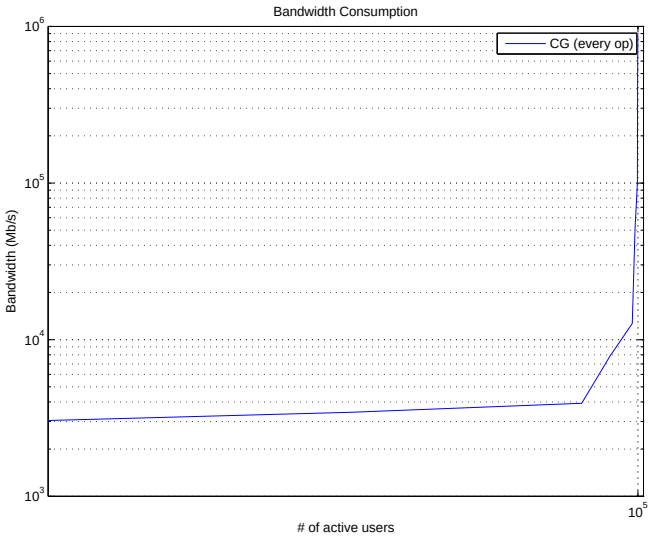

Figure 6.8: Scenario F: Bandwidth cost as the revoked member number changes

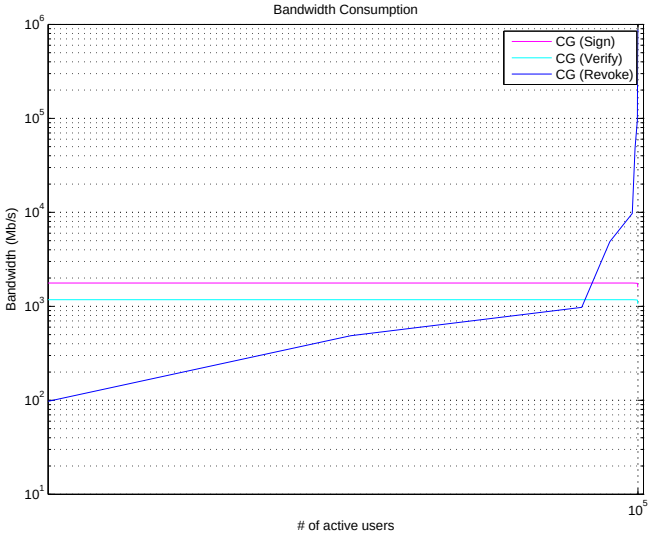

Figure 6.9: Scenario F: Bandwidth cost as the revoked member number changes (details of every operation)

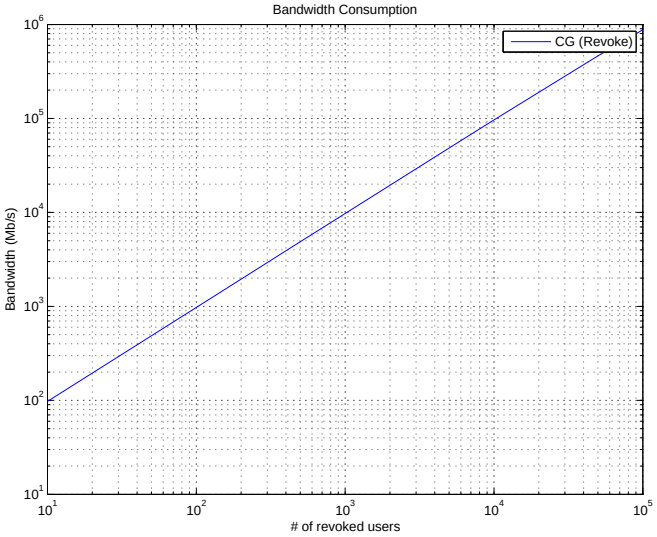

Figure 6.10: Scenario F: Bandwidth cost as the revoked member number changes, with revoked member number on the x axis

CG[a]	
Parameter	Value
l_n	1024
l_P	1024
l_Q	230
l_E	450
l_e	30
l_s	30
l_c	160
$\mathcal{H}$	SHA-1
$\mathcal{GPK}(L_{GPK})$	10470
$Signature(L_s)$	7754

[a] Full revocation version

Table 6.4: CG security parameters adopted for model evaluation

On the contrary, in terms of computation time, we have shown in the previous chapter that CG is always leading. From the results of our benchmarks we can infer that CG has a speedup of $3 - 5x$ with respect to ACJT performances for usual operations. The original paper claims an order of magnitude of improvement, thus we think it could be possible to optimize our mathematical library for better speed.

On the feature side ACJT is not so "practical": its usage is restricted to the (unlikely) case of monotonically growing or very short-lived groups. Revocation is probably not only a very sought after feature, but a "must" for real-world usage of these constructs. CG supports it but at great costs as soon as the number of revoked members in a time unit is comparable to the number of active members. We've shown that revocation taxes the bandwidth utilization with massive costs, even for a small enough percentage, for instance 1% of total group members revoked in a time unit. We can however argue that if we keep the rate of revoked members in a time unit, small to a reasonable and likely amount, say around 1% of active users, the system still performs well for many applications.

As a conclusion, ACJT scheme is quite general purpose but can suffer if the security requirements are high, it is less efficient in terms of computation time and above all does not offer revocation: for real world application these facts sound like a no go for nearly every situation. CG scheme, on the other hand, may not be practical if the number of verifiers in a protocol is high, but otherwise is the natural scheme of choice until the number of revoked members in a time unit is small enough. If a group features a very dynamic nature we can not propose a solution.

Symbol	Description	Value
L_m	Message length	8000[a]
n_s	Messages signed in a time unit	$[1:9]$
n_v	Verifiers for each signed message in a time unit	3
p_o	Percentage of opened messages	0.1%
N	Group population	1000000
p_a	Online/active group population percentage	10%
α	Members added to the group population	0
ρ	Members revoked from the group population	0[b]
N_a	Online/active group population	100000

[a] value in bits
[b] CG only

Table 6.5: Scenario B: Parameters for the test varying the number of signed messages

Symbol	Description	Value
L_m	Message length	8000[a]
n_s	Messages signed in a time unit	3
n_v	Verifiers for each signed message in a time unit	$[1:9]$
p_o	Percentage of opened messages	0.1%
N	Group population	1000000
p_a	Online/active group population percentage	10%
α	Members added to the group population	0
ρ	Members revoked from the group population	0[b]
N_a	Online/active group population	100000

[a] value in bits
[b] CG only

Table 6.6: Scenario C: Parameters for the test varying the number of verifiers for every signed message

Symbol	Description	Value
N	Group population	[100 : 5000000]
p_a	Online/active group population percentage	10%
α	Members added to the group population	0
ρ	Members revoked from the group population	0[a]
N_a	Online/active group population	[10 : 500000]

[a] CG only

Table 6.7: Scenario D: Parameters for the test varying the total member number

Symbol	Description	Value
N	Group population	1000000
p_a	Online/active group population percentage	10%
α	Members added to the group population	[0 : 100000]
ρ	Members revoked from the group population	0[a]
N_a	Online/active group population	[100000 : 110000]

[a] CG only

Table 6.8: Scenario E: Parameters for the test varying the added member number

Symbol	Description	Value
N	Group population	1000000
p_a	Online/active group population percentage	15%
α	Members added to the group population	0
ρ	Members revoked from the group population	[0 : 10000]
N_a	Online/active group population	[150000 : 148500]

Table 6.9: Scenario F: Parameters for the test varying the revoked member number

Chapter 7

Case Study: PERIMETER

In this chapter we will study the feasibility of adding group signature support to a distributed system on a 3G mobile network where a large number of users shares quality of experience feedbacks about the quality of mobile and wireless services present in an area. In our case, Quality of Experience (QoE) is a subjective measurement of the quality of a service. Every user taking part in the system has the possibility from time to time to rate subjectively the quality of a service used and to distribute it to other users upon request, together with the location where the measurement was taken. Users are then able to choose the best service based on their preferences and the requirements of the applications they are using.

7.1 PERIMETER

PERIMETER is the acronym of an ICT project funded by the European Union's seventh framework research program (FP7). Its full title is "User-Centric paradigm for Seamless Mobility in Future Internet". PERIMETER's main objective is to establish the new paradigm of user-centric strategies for advanced networking.

Nowadays mobility is already ubiquitous and more and more useful in everyday life, but it still plagued by many drawbacks: costs are never clear enough, security is often a negletted property, networks are operator-centric in the sense that subscribers are normally tied to their specific mobile operator network, even if conditions or load of that network are worse than other networks present in the vicinity. Ideally the user should not be tied to a particular network and, on the contrary, should be the in charge of choosing the best solution with respect to his preferences, like maximum cost or desidered security. PERIMETER tries to fill this gap establishing a new user-centric paradigm for advanced network-

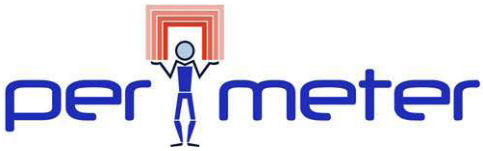

Figure 7.1: PERIMETER Logo

ing: mobile users are "Always Best Connected" (ABC), where "best" is defined by the user's preferences.

This objective is achieved by selecting the best network based on Quality of Experience (QoE), user preferences and high-level rules instead of individual technologies. In fact, every user rates his experience with a network service (3G network, WLAN, WiFi, ecc) in terms of QoE, that is a subjective measurement of the quality of available communication services as perceived by the user. This approach is richer than one purely based on technical measurements, since it encompasses a larger scope of cases and really gives the end user a service which is the "best" for him.

Every user in an area, then, periodically shares these QoE information to better inform users that are close by, about the services present in the area and their quality. At the same time every user collects other users' measurements to better choose, if possible, the best network for his preferences.

Aside from network selection, PERIMETER is much more: to achieve this seamless mobility among networks, it is very important to change the way the user interacts with network providers in such a heterogeneous system. More dynamic ways of fast authentication, authorization, accounting (AAA) will be proposed to move forward from the actual operatic-centric model of representing users.

Privacy and Trust Last but not least come the concerns about privacy, security, trust and traceability. In an environment so dynamic and open where everyone should share his reports about his vicinity, it is vital to protect the location and the identity of a user. On the other side, it is not less important to the project to provide trust and reputation, so that the QoE information exchanged are really useful and truthful. At last, it must even be possible to trace malicious users so that abuses can be punished.

It is clear that the aforementioned requirements clash together. However as we have already seen in the previous chapters of this work, there are cryptographic primitives that grant anonimity, traceability and group trust. In fact, to solve this issue, PERIMETER will employ a group anonymous authentication infrastructure based on group signature schemes fully integrated with an anonymization infrastructure to strip, from the QoE data exchanged, any trail that could lead to a user identity.

7.1.1 XPeer

The PERIMETER architecture specifies the use of the XPeer framework. XPeer [SMGC04, XPe10] is a P2P XML database system which manages and gives access to data distributed over an arbitrary set of *Peers*. The Peers are autonomous in the sense that they are free to share data of their choice and to connect or disconnect at any time.

The main advantages of XPeer are:

Zero-Administration a collection of algorithms self-organizes the SuperPeer network and allows arbitrary changes in the network topology;

Open-Schema a peer may export any kind of data provided that data are encoded in XML format and described by a schema;

Querable a basic property of a database is of course the possibility to satify query requests. XPeer adopts a subset of XQuery language.

We note that due to the the everchanging topology of the system, XPeer offers no guarantee about the completeness of query results.

Topology XPeer is a hybrid p2p system composed by a dynamic set of autonomous peers which shares data and executes queries. Some nodes (usually with adeguate computational power and resources) may perform administrative tasks for the network: they are called *SuperPeers*. Normal Peers may become SuperPeers on a voluntary basis, while retaining their Peer role.

The system is nor a pure P2P system nor only a hierarchical one. SuperPeers are organized to form a tree, where each node manages and hosts information about its children. Peers are logically organized into *clusters* of nodes, where each cluster contains at least one SuperPeer in charge of managing the cluster itself. SuperPeers having the same father form a *group*.

Since the topology of the network evolves over time, SuperPeers may change their tree structure: they may split or merge clusters and groups or promote Peers to SuperPeer status. In particular, if the workload for a given SuperPeer is too high, it first tries to relocate some Peers in other clusters (*network balancing*), then, if the problem persist, it asks the system for new SuperPeers to delegate to them part of its workload (*network extension*). If the workload is still too heavy, as a last resort, it may disconnect some Peers from the network (*peer de-gnoming*). On the other hand, if the workload is too light, the SuperPeer may decide to import Peers from other clusters or to move its children to another SuperPeer and then to abandon the SuperPeer status (*network contraction*). These algorithms together achieve the zero-administration property.

Data model XPeer poses no constraints to the type of data that can be shared as long as they are encoded in XML format and described by a schema. No global schema is enforced, so data is represented as an unordered forest of trees. Each tree is then augmented with the label of the hosting peer (location). Data are exported in the form of a *tree-guide*, that is a tree-like description of the XML document, and then searched with a tree search algorithm. The middle nodes of the tree follow the structure of the document itself, while the leaves may even contain value ranges of the data being shared.

We have already seen that Peers are clustered together: when it is possible, clusters are formed on a schema-similarity basis, that is Peers exporting data with similar schemas. SuperPeers store two kinds of schema information about their children: the list of all schemas (the *schema list*) and the union of these schemas (the *superpeer schema*). The list is used to answer to queries, while the second is passed upwards in the tree so that the schema description of a cluster is published at the top of the tree (*tree guide propagation*).

The process of querying the system is split in two phases: compilation and execution. During the former a Peer writes a query using XQuery language and sends it to the SuperPeer of its cluster. The compilation is made by traversing the tree hierarchically, till the root of the tree is reached, with a tree-search algorithm until one or more locations for the data being asked are found or none at all is found. If the search is successful, the Peer gets a list of all the locations that store data relevant for the search. In the second phase, the execution, the

Peer asks directly every Peer in the result list for data. This dual-part approach allows the queries to be compiled, optimized, and run only over a set of relevant peers, thereby avoiding broadcast.

A cluster is not only useful for smoothing the search process. In fact inside a cluster, the system may decide to replicate, partially or totally, the data being shared to balance the workload or to add redundancy. Like the superpeer status, replication too happens on a voluntary basis.

7.2 Model of a Distributed QoE Storage System on a 3G Mobile Network

7.2.1 Description

In this section we describe a simple but meaningful scenario of a distributed storage system working in a 3G mobile network using PERIMETER and XPeer. We then model and analyze the scenario and then make a comparison between the resource requirement of the system with and without the use of group signatures. Our effort is not directed on how to modify existing protocols for the use of group signatures but only to give a rough estimate of the bandwidth requirement of this scenario, so that it is possible to show its feasibility.

We suppose a 3G mobile network where the users are uniformely distributed in a circle of diameter D. This event is highly unlikely to happen in reality but our aim is to model 3G cells that manage an equal number of users each. While in this circle, users are randomly moving of uniform motion. Each user partecipating in the network exchanges Quality of Experience (QoE) data. Furthermore, each user is interested in QoE data relevant only for his actual location, both past and present, produced by other users in the same geographic area. With this information, the user is then able to decide which connection is best suited for the application he is running and the preferences set, e.g. price or reliability. We define this area of relevance as a circle of radius d and the user's location as the centre of this smaller circle.

We recognize that both our network and mobility models are simplified, but we think that on the global scale it will be indifferent which model we will employ. Further studies on this issue will be probably be carried on in the future.

In our case study every user owns a terminal with networking capabilities, which is a node running a Peer instance of XPeer. Every time a user powers up a terminal, he connects to a SuperPeer and gets access to the distributed storage system. Similarly he disconnects when he powers down the terminal. Every time a user experiences a network service he is asked to give a feedback of the quality, thus producing a QoE measurement. In addition, periodically a user polls the storage system to obtain relevant QoE measurements.

We imagine for our analysis a very simple XPeer scenario, where one SuperPeer acts as *Root* of the system and of the tree hierarchy. All other SuperPeers are in the same group (that is, their father is Root) and all Peers are clustered around them. In accordance with our model, both Peers and SuperPeers are uniformely distributed in the network. Root physical location in the network is indifferent. XPeer voluntary services for becoming a SuperPeer or for generating replicas are all disabled by default. XPeer topology for our scenario can be

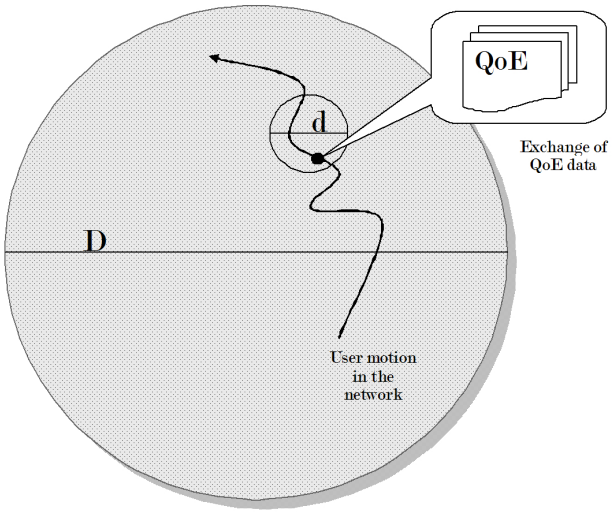

Figure 7.2: 3G QoE Network model adopted

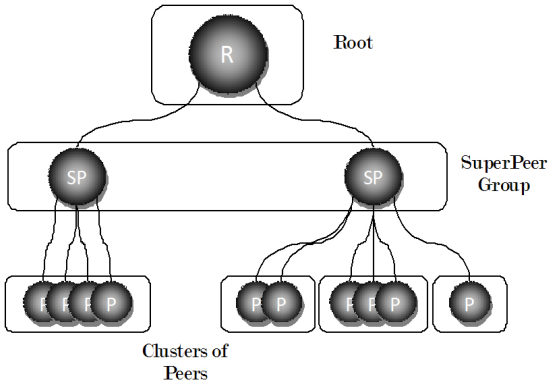

Figure 7.3: XPeer scenario topology

found in figure 7.3.

We will now present the bare model and improve it with the addition of a group signature scheme, then we will evaluate it using data coming from the 3G network world [TSSP08].

7.2.2 Definitions

In table 7.1, we briefly report definitions, useful for the understanding of the following model.

7.2.3 Model

In this section we will try to estimate the average number of exchanged messages for typical actions that a XPeer node performs. We list them in table 7.2, giving

Token	Description
P	Peer
SP	SuperPeer
R	Root
K	# of SP whose father is R
D	Network diameter
$d(\leq D)$	Network portion diameter
N	Total number of users
M_1	XPeer signalling message (<1k)
M_2	XPeer small meta-data message (<10k)
M_3	XPeer big meta-data message (>10k)
T	Seconds between two sebsequent writes
$n = N \cdot (d/D)^2$	Number of users in a geographic area

Table 7.1: XPeer on a 3G network model definitions

special attention to what they represent in our case study. For our estimate we suppose that the number of peers in a limited area is always much bigger that the number of available SPs.

XPeer action	In our Scenario	# of messages
P connection to a SP	User powers up the terminal	$4 \cdot M_1$
P disconnection from the network	User powers down the terminal	$4 \cdot M_1$
P publishing of data in the storage[a]	User gives a QoE report	$2 \cdot M_1 + 2 \cdot M_3$
P query compilation	User asks for QoE data	$2 \cdot (K+1) \cdot M_1$[b]
P query execution		$n \cdot M_1 + n \cdot M_2$

[a] Tree guide propagation.
[b] Under the hypothesis that $n >> K$.

Table 7.2: Average number of messages exchanged during typical XPeer actions

Instead, in table 7.3 we value the number of actions performed in every limited area every second, using the statistical data taken from [TSSP08]. As we have already noted the users are uniformely distributed in the network circle, thus for the number of logins and logouts we divide the network operator's average found in [TSSP08] by the ratio between the limited area and the total network area. The number of writes is just related to the frequency with which users publish QoE reports to the network. Regarding the number of reads, as we are supposing that users move of randomly uniform motion ($30Km/h$ in the original paper), their permanence in an area will be very time-limited and every user changing area polls the system for QoE data of the new area. This leads to the result that the number of reads is related to the speed with which users move in the network.

Action	Value[a]
# of logins/logouts	$694 \cdot (d/D)^2$
# of writes	$\frac{1}{T} \cdot n$
# of reads[b]	$8,3 \cdot n$

[a] for every second in a cell.
[b] in state of uniform motion $30Km/h = 8,3m/s$.

Table 7.3: 3G network statistics

7.3 Evaluation

In this section we will evaluate the models defined for the group signature schemes already seen together with this particular scenario. We give values to the parameters already seen and add the ones missing in table 7.4 and then we will graph the results, applying the values to the models. For group signatures we suppose that every mobile user belongs to the same group signature group, because we are interested in the study of the performance of our system in the whole network of a 3G operator. For security parameters we reuse the ones already precalculated with 1024 bits equivalent security. We refer the reader to the previous chapter for details on the modeling of group signature lenghts.

To complete our model we still need the values reagarding the fluctuation of the population of mobile network operators. Such data is however considered a trade secret and it is not of public domain. For this reason it is up to us to make hypotesis about reasonable values. We suppose that on the long run the population will mantain its level, so the rate new customers sign contracts or leave the operator are the same. We call this rate, *recycle rate*. We make three hypotesis about this rate in table 7.5: small, medium and high.

The results of the evaluation of our model are resumed in table 7.6. We report just one column, since the results are equal till the second decimal digit.

7.4 Analysis

The evaluation of the model exhibits the very pratical nature of protecting PERIMETER with group signatures. We showed that even CG's revocation is achieved quite easily with little overhead more than the case with ACJT. We underline the very conservative nature of our approach: forcing users to rate QoE every minute is a really inflated estimate.

We know from our previous analysis that the key parameter for the feasibility of the system is the revocation rate. We get here an indirect verification about the validity of the conclusions of the previous chapter: we concluded that with CG the system collapses as soon as the number of revoked members in a time unit is comparable to the number of active members and. This fact is, however, far from reality in our scenario. Tentatively, we found out that the ovscusaerhead of the system goes over 50% when the revocation rate is around 7000 users/s, but this would mean an annual recycle rate of around 74000%. This

Symbol	Description	Value
M_1	Message length	$1Kb$
M_2	Message length	$10Kb$
M_3	Message length	$20Kb$
K	Number of SPs whose father is Root	100
T	Seconds between two subsequent writes	60
tnotea		
D	Network area diameter	$1000Km$
d	Limited network area diameter	$1Km$
L_m	Average message length	$235Kb$[b]
n_s	Messages signed in a time unit	$0,05$
n_v	Verifiers for each signed message in a time unit	498
p_o	Percentage of opened messages	1%
N	Group population	30000000
p_a	Online/active group population percentage	10%
N_a	Online/active group population	3000000
n	Online/active members in the limited area	3

[a] That is one write every minute. This is very unlikely value, but we want our analysis to be strictly conservative.

[b] This average is the average weighted of the length of the messages on the number of actions.

Table 7.4: Values assigned to parameters for model evaluation

rate is obviously disproportionated with respect to the bounds and requirements imposed by the scenario.

The system presents no other visible limits since all relations in the model are linear.

		Recycle Rates		
Symbol	Description	Small (5%)	Medium (20%)	High (50%)
α	Members added	$0,05$[a]	$0,20$	$0,50$
ρ[b]	Members revoked	$0,05$[a]	$0,20$	$0,50$

[a] with $N = 3000000$, $x\%/\text{year} \simeq 0, x/s$.
[b] CG only.

Table 7.5: Suggested values for the mobile operator recycle rate

Security	Bandwidth[a]	Overhead
No Security	$0,73$	
ACJT GS	$0,78$	$5,91\%$
CG GS	$0,79$	$7,18\%$

[a] in Mb/s.

Table 7.6: Results of the evaluation of our model for the limited area with and without the protection of group signatures

Chapter 8

Conclusions and Future Work

During this work we reached two main original results.

The first and probably more important achievement was to model, analyze and evaluate PERIMETER's storage system, a distributed storage system running on 3G mobile network nodes, with and without the protection of group signatures. We theoretically demonstrate that, with the requirements imposed by our scenario, the overhead imposed by group signatures is acceptable and in addition that a tradeoff among security, privacy and scalability is not necessary; in fact, we accomplished all three together. In particular, we obtained *privacy* intended as the anonymization of traffic source identities and *security* as a basic user trust mechanism, through the use of group identification. We even proved *scalability* using the system in a highly populated peer-to-peer environment and *traceability* as a way to retrieve a malicious user identity. Moreover, we drew the limits of a similar system in a general case, with a wider scope.

At the same time, we showed the feasibility of designing and implementing a cryptographic framework for group signatures. Such framework exhibits many useful properties: application and implementation independence together with algorithm extensibility. Furthermore, we integrated it in Sun's existing crypotographic architecture showing that is possible to bind together modern and bleeding-edge cryptography concepts. In addition, we merged into the framework the implementation of the two group signature schemes reviewed during our work.

In the near future, we have already planned to go further one step in the natural direction. After the implementation of the framework and the scenario analysis, we will integrate the first with the running system to improve privacy and security of PERIMETER. Finally, at the same time, we should develop and analyze different and more thorough models for network topology and mobility to cover a broader range of situations that could happen in reality.

Appendix A

Glossary

We report here the common acronyms used throughout this work.

ACJT Group Signature Scheme from Ateniese, Camenisch, Joye and Tsudik [ACJT00]

API Application Programming Interface

CG Group Signature Scheme from Camenisch and Groth [CG04]

DHP Diffie-Hellman Problem

DDH Decisional Diffie-Hellman

DLP Discrete Logarithm Problem

FACTORING Integer Factorization Problem

GM Group Manager

$\mathcal{GPK}$ Group Public Key

GS Group Signature

$\mathcal{GSK}$ Group Manager Secret Key

GM Group Manager

JCA Java Cryptography Architecture

JVM Java Virtual Machine

MRL Member Revocation List

$\mathcal{MSK}$ Member Secret Key

P2P Peer-to-Peer Network

QoE Quality of Experience

SP XPeer SuperPeer

SPI Service Provider Interface

SRSA Strong RSA

VLR Verifier-Local Revocation

Appendix B

Number Theoretic Problems

Cryptography provides many different low-level algorithms called *cryptographic primitives* (i.e. encryption and description schemes or hash functions). Each of them offers precise services (i.e. authenticity or confidentiality) guaranteeing security under different assumptions. These primitives can be combined to work together to form *cryptosystems*.

The security of these systems, such as group signature schemes, relies on the intractability of solving some number theoretic problems. That means that no algorithm is known to solve those problems using a reasonable amount of resources (time and/or memory), because it is proved or assumed that no such algorithm exists. In the following we present those problems and assumptions useful to understand the group signature schemes later discussed.

B.1 Factorization of Large Integers

Efficient factorization of large integers is probably the most famous and studied number theoretic problem. Many modern cryptosystems are based on its intractability.

Definition B.1. *(Integer Factorization Problem) The* integer factorization problem *(FACTORING) is the following: given a positive integer n, find its prime factorization, so that $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, with p_i pairwise distinct primes and $e_i \geq 1$.*

As of today the best published running time of an algorithm, the *General Number Field Sieve*, solves this problem in $O(e^b)$, where b is bit length of the number to factor. No polynomial time algorithm that solves FACTORING is known, therefore for large enough b, the problem is still intractable.

On the contrary the problem of deciding if a number is prime or composite, the *primality test*, is much easier than factoring the number itself. In fact a deterministic polynomial time algorithm exists, the *AKS primality test*, that runs in $O(\log n)$. This result is of great importance because it makes possible the setup phase for the RSA problem that we describe in the following.

B.1.1 RSA Problem

The intractability of this problem is the base for the security of the RSA public key encription scheme and the RSA digital signature scheme [Lab00].

Definition B.2. *(RSA Problem) Given a positive integer n, product of two distinct odd primes p and q, choose a positive integer $e < n$ and coprime with $\phi(n) = (p-1)(q-1)$. Given an integer $c < n$, the* RSA problem *is the following: find m so that $m^e \equiv c \bmod n$.*

This task is the same as finding the e^{th} roots of a number modulo a composite number n, whose factors are not known. The constraints imposed on e and c ensure that there is exactly one m that solves the problem. As of today the most efficient mean that solves the RSA problem involves the factorization of the modulus n. According to this fact it is conjectured that the RSA and the integer factorization problem are computationally equivalent, that is for large enough n, the RSA problem is intractable.

Following the work of [BP97, FO97] an alternative stronger definition of the problem has been proposed, the so called strong RSA problem. In this formulation the exponent e is not given, but should be self-determined.

Definition B.3. *(Strong RSA Problem) Given a RSA modulus n and an integer $c < n$, the* strong RSA problem *(SRSA) is the following: find m and $e > 1$ so that $m^e \equiv c \bmod n$.*

Assumption B.1. *(Strong RSA Assumption) The strong RSA problem is intractable by every polinomial time algorithm.*

B.2 Discrete Logarithms

Another building block of classic cryptography is the discrete logarithm problem, DLP. It is most notably used in the Diffie-Hellman key exchange, in the Elgamal encryption system and in the Digital Signature Algorithm. More recently has been explited in elliptic curve cryptography. In this section we describe formally the problem.

Definition B.4. *(Discrete Logarithm) Given a finite cyclic group G and a generator of G, $g \in G$, so that every element of G can be written as a power of g. The* discrete logarithm *of an element $a \in G$, is the unique integer x, $0 \leq x < |G|$ so that $a = g^x$.*

Definition B.5. *(Discrete Logarithm Problem) The* discrete logarithm problem *(DLP) is the following: given a finite cyclic group G, a generator $g \in G$ and an element $a \in G$, find the integer x, $0 \leq x < |G|$ so that $a = g^x$ holds.*

The hardness of this problem depends strongly on the structure of the finite group involved. In some cases, in any group for which multiplication is efficent, exponentiation is also efficient. Thus there exist sub-exponential time algorithms that solve DLP. For the construction of a cryptosystem we need a stronger assumption of the hardness of the DLP.

B.2.1 Diffie-Hellman Problem

This number theoretic problem is closely related to the DLP.

Definition B.6. *(Diffie-Hellman Problem) The* Diffie-Hellman problem *(DHP) is the following: given a finite cyclic group G, a generator $g \in G$ and two elements g^u and g^v, find the element g^{uv}.*

This problem may appear in two different "flavours": Computational and Decisional. The latter is of interested for our work.

Definition B.7. *(Decisional Diffie-Hellman Problem) The* decisional Diffie-Hellman problem *(DDH) is the following: given a finite cyclic group G, a generator $g \in G$ and three elements g^u, g^v and g^w, decide if g^w and g^{uv} are equal.*

Assumption B.2. *(Decisional Diffie-Hellman Assumption) The decisional Diffie-Hellman problem is intractable by every polinomial time algorithm.*

B.2.2 Quadratic Residuosity

Definition B.8. *(Quadratic Residue) Let $a \in \mathbb{Z}_n^*$. a is a* quadratic residue modulo n *(QR_n) if there exists an $x \in \mathbb{Z}_n^*$, so that $x^2 \equiv_n a$, otherwise is said quadratic non-residue modulo n.*

For every finite group with order a odd prime p with a generator g, $|QR_p| = (p-1)/2$ and $QR_p = \{g^0, g^2, g^4, \ldots, g^{2(p-2)}\}$. In addition, there is an algorithm polynomial in the number of digits of a, that decide if an element is in QR_p. More generally, this is always possible every time the factorization of the modulus is known, thanks to the *Jacobi/Legendre symbols*. From this fact it follows that if we take $G = \mathbb{Z}_n^*$, it has been proved that the DDH does not hold in the finite group G. Still, the group of quadratic residues is itself a finite cyclic group and if g is a generator for G, $g^2 \bmod n$ is a generator of QR_n. It is conjectured that the DDH holds in QR_n, if n is a prime in the form $n = 2p + 1$ with p still prime and g a generator.

Definition B.9. *(Safe Prime) A number p is a* safe prime *if even q is prime and $p = 2q + 1$, that is if $p - 1$ has a large prime factor.*

This definition implies that there is one large subgroup of order q in the multiplicative group $\mathbb{Z}_p^*$: it is supposed that this fact, together with an appropriate generator, adds an additional layer of resistance against generic algorithms for discrete logarithms. Obviously if the order of this subgroup is small, it does not offer any additional security. Besides to the fact that the DDH holds in such a group, it is assumed that for large enough factor makes known attacks more efficient than generic ones. More generally, we want a large prime q to be divisor of $p - 1$, not limiting the choice to the factor two. This kind of group is called in number theory *Schnorr group*.

B.3 Identification Protocols

An identification protocol consists of a probabilistic algorithm and an interactive n-round protocol between a prover and a verifier. In the random oracle world such protocols can be turned into signature schemes using the Fiat-Shamir heuristic [FS86].

Bibliography

[ACJT00] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2000.

[ACJT09] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. Remarks on "Analysis of one popular group signature scheme" in Asiacrypt 2006. *IJACT*, 1(4):320–322, 2009.

[AST02] Giuseppe Ateniese, Dawn Song, and Gene Tsudik. Quasi-Efficient Revocation of Group Signatures. In *Financial Cryptography, 6th International Conference*, volume 2357 of *Lecture Notes in Computer Science*, pages 183–197. Springer, 2002.

[AT99] Giuseppe Ateniese and Gene Tsudik. Some Open Issues and New Directions in Group Signatures. In *Financial Cryptography 1999*, volume 1648 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1999.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short Group Signatures. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004.

[BCC04] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct Anonymous Attestation. In *Proceedings of the 11th ACM conference on Computer and Communications Security*, Conference on Computer and Communications Security. ACM, 2004.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of Group Signatures: Formal Definitions, Simplified Requirements, and a Construction Based on General Assumptions. In *Advances in Cryptology - EUROCRYPTO 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.

[BP97] Niko Barić and Birgit Pfitzman. Collision-Free Accumulators and Fail-Stop Schemes Without Trees. In *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 1997.

[Bre02] Emmanuel Bresson. *Protocol cryptographiques pour l'authentification et l'anonymat dans les groups.* PhD thesis, École Normal Supérieure, 2002.

[BS01] Emmanuel Bresson and Jacques Stern. Efficient Revocation in Group Signatures. In *Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 190–206. Springer, 2001.

[BS04] Dan Boneh and Hovav Shacham. Group Signatures with Verifier-Local Revocation. In *Proceedings of the 11th ACM conference on Computer and Communications Security*, Conference on Computer and Communications Security, pages 168–177. ACM, 2004.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhanq. Foundations of Group Signatures: The Case of Dynamic Groups. In *Topics in Cryptology - CT RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

[Cam97] Jan Camenisch. Efficient and Generalized Group Signatures. In *Advances in Cryptology - EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 465–479. Springer, 1997.

[Cam98] Jan Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem.* PhD thesis, Eidgenössische Technische Hochschule Zürich (ETH Zurich), 1998.

[Cao06] Zhengjun Cao. Analysis of One Popular Group Signature Scheme. In *Advances in Cryptology - ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 460–466. Springer, 2006.

[CG04] Jan Camenisch and Jens Groth. Group Signatures: Better Efficiency and New Theoretical Aspects. In *Security in Communication Networks, 4th International Conference*, volume 3352 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2004.

[CH02] Jan Camenisch and Els Van Herreweghen. Design and implementation of the *demix* anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, Conference on Computer and Communications Security, pages 21–30. ACM, 2002.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 101–120. Springer, 2002.

[CL03] Jan Camenisch and Anna Lysyanskaya. A Signature Scheme with Efficient Protocols. In *Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2003.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer, 2004.

[CLR92] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, 1992.

[CM98a] Jan Camenisch and Markus Michels. A Group Signature Scheme based on an RSA-variant. In *Report Series RS-98-27*. BRICS, 1998.

[CM98b] Jan Camenisch and Markus Michels. A Group Signature Scheme with Improved Efficiency. In *Advances in Cryptology - ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 1998.

[CP95] Lidong Chen and Torben P. Pedersen. New Group Signature Schemes. In *Advances in Cryptology - EUROCRYPT '94*, volume 950 of *Lecture Notes in Computer Science*, pages 171–181, 1995.

[CS97] Jan Camenisch and Markus Stadler. Efficient Group Signature Schemes for Large Groups (Extended Abstract). In *Advances in Cryptology - CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.

[CvH91] David Chaum and Eugene van Heist. Group Signatures. In *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer, 1991.

[DH76] Whitfield Diffie and Martin E. Hellman. New Direction in Cryptography. In *IEEE Transactions on Information Theory*, volume IT-22, pages 644–654, 1976.

[Elg85] Taher Elgamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Advances in Cryptology - CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1985.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge Proofs of Identity. In *Journal of Cryptology*, volume 1, pages 77–94, 1988.

[FO97] Eiichiro Fujisaki and Tatsuaki 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, 1997.

[FS86] Amos Fiat and Adi 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, 1986.

[GB08] Shafi Goldwasser and Mihir Bellare. Lecture Notes on Cryptography. `http://cseweb.ucsd.edu/~mihir/papers/gb.html`, Jul 2008.

[Ge04] He Ge. An Efficient Revocation Algorithm in Group Signatures. In *Information Security and Cryptology -ICISC 2003*, volume 2971 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2004.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[Gro04] Jens Groth. *Honest verifier zero-knowledge arguments applied.* PhD thesis, Århus Universitet, 2004.

[Hoe10] Jaap-Henk Hoepman. Revocable Privacy. `http://www.cs.ru.nl/~jhh/revocable-privacy/`, Apr 2010.

[HP06] Henrik Slot Hansen and Kristoffer Kjærvik Pagels. Analyse og implementation af fem gruppesignatursystemer. Master's thesis, Århus Universitet, Jan 2006.

[HRW01] Yih Huang, David Rine, and Xunhua Wang. A JCA-based Implementation Framework for Threshold Cryptography. In *17th Annual Computer Security Applications Conference 2001*, ACSAC, pages 85–91. IEEE Computer Society, 2001.

[Knu98] Jonathan B. Knudsen. *Java Cryptography.* O'Reilly, 1998.

[KP98] Joe Kilian and Erez Petrank. Identity Escrow. In *Advances in Cryptology - CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 169–185. Springer, 1998.

[KPW78] Seungjoo Kim, Sangjoon Park, and Dongho Won. Group Signatures for Hierarchical Multigroups. In *Information Security, First International Workshop, ISW '97*, volume 1396 of *Lecture Notes in Computer Science*, pages 273–281. Springer, 19978.

[KY04] Aggelos Kiayias and Moti Yung. Group Signatures: Provable Security, Efficient Constructions and Anonymity from Trapdoor-Holders. Cryptology ePrint Archive, Report 2004/076, 2004. `http://eprint.iacr.org/2004/076`.

[Lab00] RSA Laboratories. *RSA Laboratories's Frequently Asked Questions About Today's Cryptography, Vresion 4.1.* RSA Security Inc., 2000.

[Mic10a] Sun Microsystem. Java Cryptography Architecture Reference Guide for JavaTM Platform Standard Edition 6. `http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html`, Oct-Apr 2009-2010.

[Mic10b] Sun Microsystem. Java Platform Standard Edition 6, API Specification. `http://java.sun.com/javase/6/docs/api/`, Oct-Apr 2009-2010.

[Mic10c] Sun Microsystem. Java SE Security. `http://java.sun.com/javase/technologies/security/`, Oct-Apr 2009-2010.

[MvOV96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[Ngu05] Lan Nguyen. Accumulators from Bilinear Pairings and Applications. In *Topics in Cryptology - CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005.

[NKHF05] Toru Nakanishi, Fumiaki Kubooka, Naoto Hamada, and Nobuo Funabiki. Group Signature Schemes with Membership Revocation for Large Groups. In *Information Security and Privacy, 10th Australasian Conference*, volume 3574 of *Lecture Notes in Computer Science*, pages 2443–454. Springer, 2005.

[NSN04] Lan Nguyen and Reihaneh Safavi-Naini. Efficient and Provably Secure Trapdoor-Free Group Signature Schemes from Bilinear Pairings. In *Advances in Cryptology - ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.

[OOK91] Kazuo Ohta, Tatsuaki Okamoto, and Kenji Koyama. Membership Authentication for Hierarchical Multigroups Using the Extended Fiat-Shamir Scheme. In *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 446–457. Springer, 1991.

[otBC10] Legion of the Bouncy Castle. Bouncy Castle Crypto APIs. `http://www.bouncycastle.org/java.html`, Oct-Apr 2009-2010.

[Per10a] Perimeter project. `http://www.perimeter.eu/`, Set-Apr 2009-2010.

[Per10b] Perimeter project blog. `http://perimeter.tssg.org/`, Set-Apr 2009-2010.

[PM10] Andreas Pfitzmann and Hansen Marit. A terminology for talking about privacy by data minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management. `http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.33.pdf`, Apr 2010. v0.33.

[Sch91] Claus-Peter Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[Sho02] Victor Shoup. A Primer on Algebra and Number Theory for Computer Scientists, Sep 2002.

[SMGC04] Carlo Sartiani, Paolo Manghi, Giorgio Ghelli, and Giovanni Conforti. XPeer: A Self-Organizing XML P2P Database System. In *EDBT Workshops*, Lecture Notes in Computer Science. Springer, 2004.

[Son01] Dawn Xiadong Song. Pratical forward secure group signature schemes. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, Conference on Computer and Communications Security. ACM, 2001.

[TSSP08] Dario Tonesi, Luca Salgarelli, Yan Sun, and Thomas F. La Porta. Evaluation of Signaling Loads in 3GPP Networks. In *IEE Wireless Communications*, volume 15 of *Lecture Notes in Computer Science*, pages 92–100. IEEE, 2008.

[TW05] Mårten Trolin and Douglas Wikström. Hierarchical Group Signatures. In *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005*, volume 3580 of *Lecture Notes in Computer Science*, pages 446–458. Springer, 2005.

[TX03] Gene Tsudik and Shouhuai Xu. Accumulating Composites and Improved Group Signing. In *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 269–286. Springer, 2003.

[Wik10] Wikipedia. `http://en.wikipedia.org/`, Set-Apr 2009-2010.

[XPe10] Xpeer project. `http://www.di.unipi.it/~manghi/XPeerWeb/homexpeer.htm`, Set-Apr 2009-2010.

[ZL05] Sujing Zhou and Dongdai Lin. On Anonimity of Group Signatures. In *CIS (2)*, volume 3802 of *Lecture Notes in Computer Science*, pages 131–136. Springer, 2005.

[ZW08] Jing-Liang Zhang and Yu-Min Wang. Efficient Membership Revocation in ACJT Group Signature. In *Journal of Electronic Science and Technology of China*, volume 6 of *Lecture Notes in Computer Science*, pages 39–42. Springer, 2008.