A SECURITY AND PRIVACY INFRASTRUCTURE FOR CLOUD COMPUTING USING GROUP SIGNATURES

by

Fırat Tahaoğlu

Submitted to the Graduate School of Sabancı University
in partial fulfillment of the requirements for the degree of
Master of Science

Sabancı University

February, 2012

A SECURITY AND PRIVACY INFRASTRUCTURE FOR CLOUD COMPUTING USING GROUP SIGNATURES

Approved by:

Assoc. Prof. Dr. Albert Levi (Dissertation Supervisor)	....
Assoc. Prof. Dr. Erkay Savaş	....
Assoc. Prof. Dr. Berrin Yanıkoğlu	....
Assist. Prof. Dr. Cemal Yılmaz	....
Prof. Dr. Alev Topuzoğlu	....

Date of Approval: ....

A SECURITY AND PRIVACY INFRASTRUCTURE FOR CLOUD COMPUTING USING GROUP SIGNATURES

Fırat Tahaoğlu

Computer Science and Engineering, Master's Thesis, 2012

Thesis Supervisor: Albert Levi

Keywords: Group Signatures, Cloud Computing, Cloud Security, Access Rights.

Abstract

New software applications are being developed every day by software development groups, ranging from the most professional to smaller amateur ones. The structures of the software development groups are very diverse, and a development environment should satisfy the needs of different kinds of group structure. Considering the advantages of low resource requirement, accessibility through mobile devices with restricted resources, and compatibility with collaborative working environments, Cloud computing is a perfect match for software developers, especially for the groups. However, since Cloud computing operates on insecure Internet, security against malicious third parties is a crucial issue. Files should be kept safe in the Cloud, and should only be accessed by those who have the authorization. Revocation and addition of the group members, and the organization of the access rights should also be performed in an efficient and robust way, fulfilling the needs of different groups.

In this thesis, we propose a security and privacy infrastructure for a software development environment running in the Cloud. We propose to solve the security issues using the anonymous credential system, *idemix*, provided by IBM Research which relies on the Camenisch-Lysyanskaya group signature scheme. Group signatures can provide flexibility in the groups' inner organization and are also helpful for handling the access rights. Moreover, using an anonymous credential system also provides to the group members the ability to keep their anonymity while interacting with Cloud. In this way, we aim to provide an infrastructure to serve the groups with different inner organizations by not compromising their privacy. In order to evaluate the performance of the proposed system, we develop a simulation environment using M/D/m/m queues and analyze the proposed system under different scenarios and access control structures. Our results show that the proposed system is an efficient one and can serve up to 1000 concurrent users with response time under one second using four servers.

BULUT BİLİŞİM İÇİN GRUP İMZA YAPILARI KULLANAN BİR GÜVENLİK VE MAHREMİYET ALTYAPISI

Fırat Tahaoğlu

Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans Tezi, 2012

Thesis Supervisor: Albert Levi

Keywords: Grup İmzalari, Bulut Bilişim, Güvenlik

Özet

Yazılım geliştirme grupları tarafından her geçen gün yeni yazılımlar üretilmektedir. Bu gruplar, en amatöründen en profesyoneline, nitelik açısından çok değişik özelliklere sahip olabilmektedirler. Bir geliştirme ortamı, iç yapısından bağımsız olarak her farklı gruba hizmet edebilme özelliğini taşımak durumundadır. Bulut bilişimin sağladığı düşük işlem güçlü mobil araçlarla çalışabilme kapasitesi, yardımlaşmalı çalışma ortamlarıyla uyumu gibi kolaylıklar, bulut bilişimi yazılım geliştirme grupları için çok uygun bir alternatif haline getirmektedir. Ancak, bulut bilişimin güvensiz internet ortamında çalışması, güvenlik sorunlarını da önemli kılmaktadır. Dosyalar bulut içerisinde güvenli bir biçimde tutulmalı ve ancak erişim izni olanlar tarafından erişilebilir kılınmalıdır. Grup üyelerinin eklenmesi ve çıkarılmasının ve grup içindeki erişim izni haklarının her grubun iç organizasyonuna uyumlu olarak güvenli ve verimli olması gerekmektedir.

Bu tezde bulut içerisinde çalışan bir güvenlik ve mahremiyet altyapısı önerilmektedir. Güvenlik problemleri IBM'in geliştirdiği anonim bir kimlik sistemi olan ve Camenisch ve Lysyanskaya tarafından geliştirilmiş grup imza yapısına dayanan *idemix* kütüphanesiyle çözülmektedir. Grup imza yapıları, özellikleri itibariyle grupların iç organizasyonlarında esneklik sağlamakta ve grup içi erişim izinlerini idare etmekte oldukça yardımcı olmaktadırlar. Bunun yanısıra anonim kimlik sistemleri kullanılarak, grup üyelerinin servis sağlayıcıya karşı anonimlikleri ve dolayısıyla mahremiyetleri korunmaktadır. Böylelikle, iç yapılarında değişik özellikler barındıran gruplara mahremiyetlerine ve güvenliklerine zarar vermeden hizmet verecek bir altyapı sunulşmuştur. Sistemin verimliliğini ölçmek için, M/D/m/m kuyruk modellerini kullanarak bir simulasyon ortamı geliştirilmiştir. Bu simulasyon ortamında değişik senaryolar ve erişim hakları yapıları kullanarak sistemin verimliliği analiz edilmiştir. Sonuçlarımız, önerdiğimiz sistemin verimli bir sistem olduğunu ve 1000 eşzamanlı kullanıcıya, dört sunucuyla, bir saniyenin altında kalan cevap süreleriyle hizmet edebildiğini göstermiştir.

Acknowledgements

I would like to thank my family for their endless love and support throughout my life.

I would also like to thank my supervisor Albert Levi for his support, guidance and understanding during my research and more importantly my whole academic life.

The support of Aysu and Mr. Karakulak should not go unnoticed. I could not have done all this work without their love and incredible sense of humor.

I appreciate the valuable feedbacks of my thesis jury members.

I will never forget my time on FENS 2014 and I would like to thank all people there for being such great friends and making a regular lab a great environment.

Last, but not least, I would like to thank Esra Erdem for her guidance throughout my undergraduate years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing is a new concept, which offers delivery of computing as a service rather than a product. Cloud providers aim to deliver applications, or various services, which can be accessed from desktop or mobile applications via the Internet. There are various kinds of Cloud services that Cloud providers offer ranging from simple applications to complex infrastructures. The main motivation is to enhance mobility, collaboration and easy maintainability due to the centralization of the services. Cloud applications are getting more and more widespread, thanks to their low resource requirements, accessibility through mobile devices with restricted resources, and compatibility with collaborative working environments such as GoogleDocs [7] and Dropbox [3].

New software applications are being developed every day by software development groups, ranging from the most professional to smaller amateur ones. This diversity also shows itself in the inner hierarchies of these development groups. While there is a flat organization in some of them, some others have rigid hierarchical structures. Moreover, members of these groups may be located in different countries, and have no personal interaction. In other words, the structures of the software development groups are very diverse, and a development environment should satisfy the needs of different kinds of group structures.

Since Cloud computing operates on insecure Internet, security against malicious third parties is a crucial issue. Files should be kept safe in the Cloud, and should only be accessed by those who have the authorization. Revocation and addition of the group members, and the organization of the access rights should also be performed in a robust way, while fulfilling the needs of different groups.

What we propose in this thesis is to utilize the advantages of low resource requirement and easy accessibility of Cloud computing in order to come up with a security and privacy infrastructure for a software development environment running in the Cloud. Our main focus is to create a generic environment that can serve differently organized software development groups. For this purpose, we make use of the group signature structure introduced by Chaum and van Heyst [21], which allows us to free the development groups in their inner organization, while being useful for interacting with the Cloud securely against malicious third parties. Moreover, since we outsource the data and the computitatons to Cloud, the Cloud provider should be working in need-to-know basis. Therefore, it is essential to achieve authentication with only the information the groups want to reveal. Hence, we implement the protocols of Camenisch and Lysyanskaya Signature Scheme [16] using *idemix* primitives. Camenisch and Lysyanskaya Signature Scheme [16] relies on Zero-Knowledge proofs, which are essential to achieve mutual authentication without exchanging any critical knowledge. This way, we secure the anonimity of the users against the Cloud service provider.

To summarize, we propose a security and privacy infrastructure for a software development environment running in the Cloud. Considering the advantages of easy accessibility, high computing power and support for collaborative features, Cloud computing is a perfect match for software developers, especially for the groups. We propose to solve the security issues using group signatures, which can provide flexibility within the groups and can be modified for handling the access rights. In this way, we aim to serve the groups with different organizations: flat or hierarchical.

We also develop a simulation environment using M/D/m/m queuing model in order to evaluate the performance of the proposed system. Our results show that the proposed system is an efficient one and can serve up to 1000 concurrent users with four servers. We also analyze the system assuming the worst case, where every action requires authentication. Moreover, we also analyze the behaviour of the system in case of user revocation.

Outline of the thesis is as follows. In Chapter 2, we give an overview of Cloud computing, Cloud Security, *idemix* library and the group signature schemes it uses. We elaborate on possible access right models using the components of *idemix* with the proposed system. In Chapter 3, we take a deeper look into access right issues using *idemix* and give insight to protocols used inside the system (namely Issue Protocol, Authentication Protocol and File Transfer Protocol). Chapter 4 explains the implementation details, message structures within the protocols, the role of *idemix* library and how it was used. Chapter 5 deals with the time measures for atomic operations and discusses how the performance evaluation was achieved, in addition to presenting the results. Finally, in Chapter 6, we summarize the results, and propose some future work on the subject.

Chapter 2

Background Information

In this section, we provide background information about the underlying technologies and literature used in the proposed system. Firstly, we give the definition of Cloud computing, its characteristics and the deployment models. After that, we present the current state of art in Cloud security. Since we used the *idemix* [33] library in the proposed system, we also provide background information about it and its underlying cryptographic foundations, namely the group signatures. We firstly provide an introduction and background information about group signatures, then give the mathematical background about Camenisch-Lysyanskaya signature scheme [16], which is the building block of the *idemix* library. Lastly, we provide information about the *idemix* library, which has been used intensively in the proposed system.

2.1 Cloud Computing

Cloud computing has recently attracted a great interest from both simple users and large-scale companies. Amazon Web Services [2], GoogleAppEngine [5], GoogleDocs [7], Microsoft Azure [8], Dropbox [3] are some of the famous examples of Cloud computing, which emerged with wide use and increasing speed of Internet. It is a widely used technology for flexible on-demand access and is adopted by more and more people and companies. NIST (National Institute of Standards and Technology) defines Cloud computing as follows [31]:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

While this definition captures the meaning of Cloud computing, Armbrust et al. [10] defines Cloud computing as both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services.

Even though it is criticized for renovating existing concepts rather than offering a new concept, the main motivation behind Cloud computing is to promote flexibility and availability. The Cloud model defined in [31] consists of five essential characteristics:

- *On-demand self-service*: A user can unilaterally provision computing resources such as network storage, computing power, server time, etc.
- *Broad network access*: Capabilities of cloud services are available over the network and accessed through various platforms, such as mobile phones, laptops, etc.
- *Resource pooling*: Resources like storage, processing, memory, virtual machines are pooled to serve multiple users using a multi-tenant model, meaning that they are dynamically assigned and reassigned according to demand.
- *Rapid elasticity*: Cloud services can be rapidly provisioned and should be able to quickly scale out or in.
- *Measured Service*: Resource usage can be controlled, monitored, and reported, providing transparency for both the users and the provider.

It is widely accepted that Cloud computing is not a new concept, however it is clearly a new paradigm. Software companies offer various Cloud services ranging from data outsourcing to office software. Since there are varying types of applications, NIST also defines service models for Cloud computing paradigm.

- *Cloud Software as a Service (SaaS)*: The user is provided with the capability to use provider's applications, which run in the Cloud infrastructure. Since the software is running in the Cloud, client can use these services regardless of its devices' computing power. However, the user has limited access to the Cloud and s/he can interfere with the user-specific application configuration settings. In other words, she cannot develop her own software, but only the ones which are already permitted by the provider. GoogleDocs [7] and GMail [4] are some examples of such service model.
- *Cloud Platform as a Service (PaaS)*: The user has permission to deploy user-created or permitted applications onto the Cloud infrastructure using programming languages or tools that provider supports. However, user is not permitted to have control over the network, servers, operating systems or storage. GoogleCode [6] is an example of such a service model.

- *Cloud Infrastructure as a Service (IaaS)*: Unlike SaaS and IaaS, user has control over processing, storage, networks, and computing resources. User is permitted to run arbitrary software, operating systems, applications. The control over underlying structure is still limited. Amazon EC2 [1] is an example of such a service model.

There are also four different types of deployment models for Cloud infrastructure. If the Cloud is operated solely for an organization, it is called a *private Cloud*. If it is made available to the general public in pay-as-you-go manner, it is called *public Cloud*. NIST also defines *community Cloud*, as the Cloud infrastructure shared by several organizations, and the *hybrid Cloud* is defined as the composition of two or more Clouds. However, Ambrust et al. [10] did not refer to the latter two terms.

Since this thesis focuses on a secure software development in Cloud environment, we can define our proposed system as Cloud Software as a Service (SaaS) in any Cloud (private or public). PaaS and IaaS are out of scope of this thesis; interested reader may find more in [10] and [31]. The rest of this section discusses the advantages and motivation of SaaS.

Cloud computing has recently become a widely used technology for on-demand and collaborative access to shared and configurable computing resources. Applications like GoogleDocs and DropBox are being adopted by more people day by day. Taking advantage of many useful features of Cloud computing, these applications provide users with different alternatives for collaborative editing and file sharing.

Since Cloud computing allows us to run the costly computations on the Cloud, users only need enough processing power to be able to run a simple interface and interact with the Cloud. In the course of our project, compiling and debugging large-scale projects are considered as computationally expensive operations especially for mobile devices. The fact that these operations are to be handled in the Cloud frees the software group members from the necessity to have powerful and expensive computers. All the group members need to have is reduced to computers with Internet connection that can run the interface. Hence, group members are able to use the system via mobile devices that have very low computing resources, providing the group members with a significant level of mobility. Moreover, since many software development groups are international ones, such a centralized system based on Cloud computing is a very suitable medium for working collaboratively from distant locations.

Furthermore, Cloud computing equips us with the advantage of being able to keep all files and programs in a single virtual location. From the software development point of view, this centralized system addresses many issues per se. For instance, when an upgrade is going to be performed on the software development environment, ensuring that all users have done the upgrade is not a trivial task; some users may upgrade, some may not and this case may cause inconsistencies. Cloud computing simply solves this problem. Since upgrading the system on the Cloud is sufficient for all users to access the upgraded version, they do not need to install the patch individually. This makes the system maintenance much easier. Moreover, the fact that all files are kept on the Cloud automatically provides fault tolerance such that data loss by single user mistake becomes very unlikely.

By keeping all files in a single server, Cloud computing also enables a collaborative environment. As in the example of GoogleDocs, making it possible for several users to work on the same file at the same time and to be able to keep track of by whom a change is made, is a valuable asset for software groups that work collaboratively.

2.2 Cloud Security

Since Cloud applications run on insecure Internet they need to be secured like every client-server application. Existing literature is focused mainly on security problems for outsourcing data [24], virtual machine security [34] and security measurements [23] within Cloud against malicious users. However, to the best of our knowledge, none of them are related to a generic security and privacy infrastructure.

Chow et al. [23] focus on new emerging security threats by the rise of the Cloud computing paradigm. The authors seperate the threats into three types:

- **Traditional Security**: This category includes traditional security attacks, which became easier by moving to the Cloud. VM-Level attacks, Cloud provider vulnerabilities, phishing are identified as such kind of threats. 'Expanded network attack surface', increasing difficulty for forensic investigation in Cloud and fitting existing authentication and authorization schemes to Cloud paradigm are also pointed out as new vulnerabilities.

- **Availability**: Since Cloud computing paradigm offers a centralized system, it raises problems such as uptime, single point of failure and assurance of computational integrity.

- **Third-Party data control**: This category includes the possible security leaks of outsourced data, such as due dilligence, audability, provider espionage, etc.

Wei et al. [34] propose an antivirus approach against the possible security breaches, targeting image repositories that are used commonly for Cloud computing. The main motivation behind this is to identify the risks both the users and the administrators can face during managing the virtual-machine images that encapsulate each application in the Cloud. The authors propose an image management system called Mirage. Zhang et al. [35] focus on security threats to elastic applications and identifies security objectives that should be provided by the infrastructure. The authors propose an elastic framework in order to achieve authentication and secure communication mechanisms for weblets running on mobile devices and Cloud concurrently. Christodorescu et al. [24] argue that Cloud security cannot be degraded only to virtualization security and focuses on how to recognize malicious code fragments in Cloud and maintain security against them. For that purpose they propose a framework that enables security services for Cloud environments where users use a variety of operating systems which need to be quarantined in case of abuse. The authors develope a novel algorithm for secure introspection which they applied during identification of guest operating system and rootkit detection.

2.3 Group Signatures

Group signatures have first been proposed by Chaum and van Heyst in 1991 [21]. The main motivation behind Group signatures is to combine security and privacy. Group signature scheme is a cryptographic method that allows a group member to anonymously sign a message on behalf of the group. The main players are:

- *Group Members*, who can anonymously issue signatures.
- *Group Manager*, who has the authority to reveal the identity of the actual signer if necessary and is responsible for adding and revoking group members.

The Group signature scheme is said to be *dynamic* if new user registration and user revocation is possible.

Bresson and Stern [15] identify the security requirements of a group signature scheme. A group signature scheme allows any user (not necessarily a group member) to be able to verify that the message has been signed by an actual member of the group. However, no one, but the group manager, can identify who actually signed the message. Also the signatures should be unlinkable, that is deciding whether two signatures have been signed by the same member or not must be infeasible.

A group signature scheme consist of five basic methods as described below [15]:

- **Setup**: A probabilistic algorithm initializing public parameters and providing a secret key to the group manager.

- **Join**: An interactive protocol between the group manager and a new user to become a new group member.
- **Sign**: The cryptographic operation to compute a group signature share using a message and a member's secret share.
- **Verify**: An algorithm, which is run by any user, in order to check whether or not a signature has been produced by an authorized signer.
- **Open**: An algorithm allowing the group manager to obtain the identity of the member who actually signed a given message.

Main properties of a group signature scheme are summarized as follows [15], [17]:

- **Correctness**: Any signature generated by a group member should be valid.
- **Unforgeability of signatures**: Only group members are able to sign messages. Furthermore, they must only be able to sign in such a way that, when the signature is presented to the group manager, he will be able to reveal the identity of the signer.
- **Anonymity of signatures**: It is not feasible to find out the group member who signed a message without knowing the group managers secret key.
- **Unlinkability of signatures**: It is infeasible to decide whether two signatures have been issued by the same group member or not.
- **No framing**: No coalition of the members (including the group manager) can sign on behalf of a non-involved group member.
- **Unforgeability of tracing verification**: The manager cannot accuse a signer falsely of having originated a given signature.
- **Coalition resistance**: No coalition of members can prevent a group signature from being opened.

There are several group signature schemes in the literature. Chaum and van Heyst [21] propose four different schemes. Chen and Pedersen propose two new group signature schemes [22], based on undeniable signatures [20]. However, some of them are not dynamic, and all of them are relatively innefficient, due to the reason that the signature size grows linearly with respect to group size. Camenisch and Stadler propose a scheme [18], which provided a constant size signature and a constant size public key. The scheme is enhanced by Bresson and Stern by adding member revocation [15]. Ateniese et al. [11] propose an interactive and coalition-resistant scheme with enhanced efficiency . Camenish and Lysyanskaya generalize this scheme and propose the Camenish and Lysyanskaya Signature Scheme [16], which is the building block of the *idemix* library. Since

CL (Camenish and Lysyanskaya) signature scheme relies on zero-knowledge proofs, we provide preliminaries for this topic below. Later we will give the details of CL signature scheme.

2.3.1 Zero Knowledge Proofs

Zero-knowledge proofs were first defined in a draft of “The Knowledge Complexity of Interactive Proof-Systems” by Goldwasser et al. [29] as the proofs that reveal no additional knowledge other than the correctness of the proposition in question. This should be the case even if the verifier does not follow the protocol but tries to cheat and trick the prover to reveal some information. A zero-knowledge proof should satisfy the following three properties:

- **Completeness:** Any true statement can be proven.
- **Soundness:** Any false statement cannot be proven.
- **Zero-knowledge:** Any verifier does not learn anything except that a statement is true.

Informally, a zero-knowledge proof can be explained with a well-known story by Jean-Jacques Quisquater [32]. Peggy (the prover) and Victor (the verifier) are near a circle-shaped cave, in which the passage from one end to the other is blocked by a magic door. Peggy claims to know the secret word that opens the door but Victor says he will not pay for the word without being sure that Peggy knows it. Peggy, on the other hand, refuses to tell the secret word before receiving the money. They decide to follow a method that will prove to Victor that Peggy knows the secret word without revealing the word to Victor. This is a zero-knowledge proof.

First, Peggy goes in the circle-shaped cave without Victor seeing from which end she entered. Then, Victor chooses randomly the side he wants Peggy to come out of and shouts out the name of this side. If it is the same end Peggy entered by, she will be able to come back no matter she knows the secret word or not, since she does not need to go through the magic door, if she is already in the side that Victor shouted. However, if it is the opposite end, she can come out from it only if she knows the secret word. If they repeat this process many times, and if Peggy does not know the secret word, the probability that she would manage to come out from the desired end would be very small. Thus, if Peggy repeatedly comes out from the end that Victor shouts, it is very likely that she knows the secret word. Figure 2.1 illustrates the story.

Hence, zero-knowledge proofs are probabilistic rather than deterministic, since there is a small probability, called the soundness error, that the verifier can be convinced on

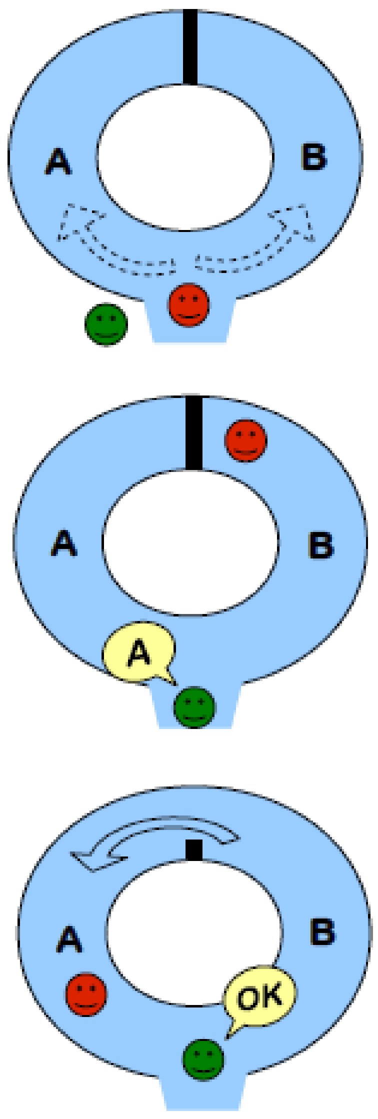


Figure 2.1: Zero Knowledge Proofs

the correctness of a false statement. However, the soundness error can be decreased to negligibly small values.

Goldwasser et al. [29] give zero-knowledge proof systems for the languages of quadratic residuosity and quadratic nonresiduosity. In number theory, an integer x is called a quadratic residue modulo m if there exists an integer x such that $x^2 \equiv y \mod m$ The quadratic residue problem with parameters $m \in N$ and $x \in Z_m^*$ consists of computing $Q_m(x)$, which is 0 if x is a quadratic residue $\mod$ m, 1 otherwise. If one knows the factorization of m, $Q_m(x)$ is easy to compute, otherwise it is infeasible.

Moreover, Goldreich et al. [28] showed that a zero-knowledge proof system for the NP-complete graph coloring problem with three colors can be created, assuming unbreakable encryption exists. This means that under this assumption, zero-knowledge proofs can be created for all problems that can be solved in polynomial time in a non-deterministic Turing machine (NP), since any problem in NP can be efficiently reduced to the graph coloring problem [28].

Goldreich et al. [12] went further to show that, under the same assumption, any problem that can be proved by an interactive proof system can be proved with zero knowledge.

Ben-or et al. [13] show that using multiple independent provers that allow the verifier to examine the provers in isolation to avoid being misled, all problems in NP can be shown to have zero-knowledge proofs without need for any intractability assumptions.

Dwork et al. [25] initiated the research on concurrent zero-knowledge proofs, which can be used in an Internet-like setting. To this end, witness-indistinguishable protocols have been proposed, which are like zero-knowledge proofs but not as problematic with concurrent execution of multiple protocols [26].

We have used the *idemix* [33] library intensively during the implementation. Now, we are going to provide some notation that will be used to describe the underlying cryptographic foundation behind the *idemix* library, namely the Camenish-Lysyanskaya (CL) signature scheme [16].

- Let S be a set. $\#S$ denotes the number of elements in the set S, and $x \in_R S$ means that the element x is chosen randomly with uniform density from S.
- The concatenation of numbers or strings is denoted by the operator $||$.
- Let $H : \{0,1\}^* \to \{0,1\}^{l_H}$ be a hash function. $\pm\{0,1\}^l$ denotes the set of integers $\{-2^l + 1, ..., 2^l - 1\}$, and $\{0,1\}^l$ stands for the non negative part of the same set, i.e. $\{0, ..., 2^l - 1\}$. Note that the current *idemix* implementation uses SHA-256 [9] hash function.

The notation of Camenisch and Stadler [18] is used for zero-knowledge proofs when presenting protocols. To give an example,

$$PK\{(\alpha,\beta,\gamma) :\ y = g^\alpha h^\beta \ \wedge\ \tilde{y} = \tilde{g}^\alpha \tilde{h}^\beta \ \wedge\ (v < \alpha < u)\}$$

denotes a zero-knowledge Proof of Knowledge of integers α, β, γ satisfying $y = g^\alpha h^\beta$ and $\tilde{y} = \tilde{g}^\alpha \tilde{h}^\beta$ with $v < \alpha < u$, where $y, g, h, \tilde{y}, \tilde{g}, \tilde{h}$ are elements of some groups

$G = \langle g \rangle = \langle h \rangle$ and $\tilde{G} = \langle \tilde{g} \rangle = \left\langle \tilde{h} \right\rangle$. All parameters except for the Greek letters, which denote the attributes to be proved, are known by the verifier. Only prime order groups are used in the *idemix* library [33]. It is well known that in such groups there exists a knowledge extractor which can extract these attributes denoted by Greek letters from a successful prover.
In the *idemix* library [33], all zero-knowledge proofs are implemented as a common three-move zero-knowledge protocol, made non-interactive using the Fiat-Shamir heuristic [27]. The values of the form $t = g^r$ used in the first flow of the protocol will be called t-values, while the responses computed in the third flow of the form $s = r - c^{\alpha}$ will be referred to as s-values. The challenge, c, has the hash of the t-values, common inputs, and also a common string called the context string, consisting of a list of all public parameters and the issuer public key. This way, the values generated during the proof cannot be re-used in any other context.

2.3.2 The CL Signature Scheme

The CL signature scheme relies on zero-knowledge proofs [16]. Using general zero-knowledge proofs, it is possible to prove statements such as “I have a signature” without saying anything more than that. Using this property, the groups have to show no more than they want, and a verifier can verify the signatures without the need to know any more than the signer wants to show. Hence, anonymity and security against Cloud is accomplished using such a structure. The main building block of the *idemix* library is the CL Signature scheme [16]. The main definitions and protocols of this scheme are as follows:

- **Key generation:** On input ℓ_n, choose an ℓ_n-bit RSA modulus n such that $n \leftarrow pq,\ p \leftarrow 2p' + 1,\ q \leftarrow 2q' + 1$, where p, q, q' are primes. Choose uniformly at random, $R_0, ..., R_{L-1}, S, Z \in QR_n$ (quadratic residues $\mod n$). Output the public key $(n, R_0, ..., R_{L-1}, S, Z)$ and the secret key p.

- **Message space:** Let ℓ_m be the size of attributes. The message space is the set

$$\{(m_0, ..., m_{L-1}) :\ m_i \in \pm\{0,1\}^{\ell_m}\}$$

- **Signing algorithm:** On input $m_0, ..., m_{L-1}$, choose a random prime number e of length $\ell_e > \ell_m + 2$ and a random number v of length $\ell_v \leftarrow \ell_n + \ell_m + \ell_r$, where ℓ_r is the security parameter required in the proof of security of the credential system. Compute the ordered set of attributes A such that

$$A \leftarrow \left(\frac{Z}{R_0^{m_0} ... R_{L-1}^{m_{L-1}} S^v}\right)^{1/e} \mod n$$

The signature on the message $(m_0, ..., m_{L-1})$ consists of (A, e, v).

- **Verification algorithm:** To verify that the tuple (A, e, v) is a signature on message $(m_0, ..., m_{L-1})$, check that

$$\equiv A^e R_0^{m_0} ... R_{L-1}^{m_{L-1}} S^v (\text{mod}\, n)\ ,\ m_i \in \pm\{0,1\}^{\ell_m},\ and\ \ 2^{\ell_e} > e > 2^{\ell_e - 1}$$

all hold.

2.4 The *idemix* Library

idemix was developed at IBM Research Zurich by a research group under the leadership of Jan Camenisch [19]. It is an anonymous credential system, supporting accountability of transactions by performing anonymous authentication between users and/or service providers.

The issuer issues the *idemix* credential in accordance with the user's attributes, which allows for various protocols such as property proofs, revocation of credentials, verifiable encryption, etc. The credential issuance and the show proof protocol are the main protocols performed. Camenisch-Lysyanskaya signature scheme [16] is used in these protocols, as it is the building block of the *idemix*. Therefore, for the successful execution of these protocols, all participants need to be in the same group and share the same system parameters, which define, for example, the size of the RSA modulus, or the domain of a hash function.

The user and the issuer create a credential interactively during the issuance protocol. To make the credential easily verifiable using the issuer's public key, the issuer signs the resulting credential with its private key. The user's master secret is bound to the credential by embedding the user's pseudonym in it [33]. Figure 2.2 illustrates a use-case diagram of the main protocols of *idemix*.

Solutions based on *idemix* send only proofs, providing unlinkable disclosure of selective credential attributes without revealing others. The user can decide on which attributes of her digital face to disclose and which to hide herself since she is involved in the disclosure process directly [14]. When the user needs to convince another entity that she possesses an attribute signed by the issuer, several zero-knowledge proofs are performed, so that the credential itself is never revealed. Hence, the same credential can be shown repeatedly without the other entity being able to link the information [33].

idemix consists of several components. The most important ones are:

- **System Setup:** As mentioned before, all participants must share the same system parameters to successfully perform cryptographic protocols. These system parameters are defined in two connected files, the SystemParameters file and the GroupParameters file.

- **Attributes:** Attributes can be name, surname, affiliation etc. Formally, they are tuples, each unique in its scope, consisting of name, value and type $(a = n, v, t)$. Integer, String, Date and Enumerations are the types of attributes that are allowed. While proving a set of attributes, user can specify which to reveal and which to hide to verifier. The attributes can contain all the information about a user. However, user may only need to show her birth date in a proof. This can be specified in a ProofSpec. An example of a ProofSpec will be given in the Chapter 4.

- **Credentials:** A credential is a certificate attesting to some attributes the user possesses. It consists of a set of attributes together with some cryptographic information. The credential is obtained from the issuer using the issuance protocol. It is used in the computation of statements about the value of the attributes it contains, when this needs to be proved to another entity. The credential itself, however, is never revealed.

For implementing the functionalities of the Camenisch - Lysyanskaya group signature scheme, we use the *idemix* primitives. We also use the XML structures for nonces, proofs and credentials, which can be processed by the *parser* class provided by the *idemix* library.

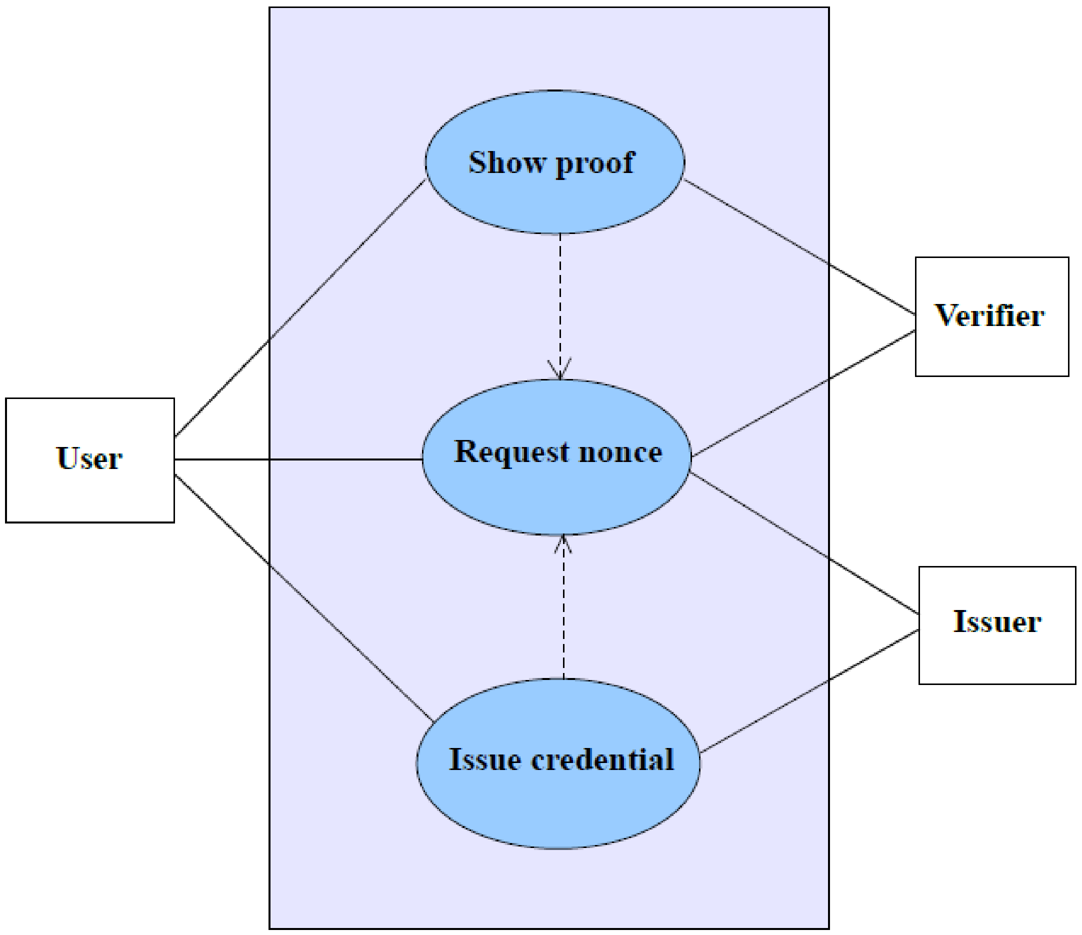


Figure 2.2: *idemix* Overview

Chapter 3

Proposed System

This section deals with the general overview of the proposed system in this thesis. The main entities in the system are groups (consisting of users of different roles), the group manager (issuer), the verifier and the server. The verifier can be considered as a module in the server, since the server starts to interact with users after the verifier authenticates them. The issuer operates in Cloud, like the verifier and the server, but as a different entity. This frees the verifier and the server to have the responsibility and knowledge about group structures and users' details. Groups are software development groups consisting of users with different roles, which can be defined during the group registration. Each group can have different roles with different hierarchical structures. The issuer also operates as the group manager, in the sense that it is responsible for user registration / revocation. It interacts with the server after group registration to inform the server about the access right scheme, which defines the roles and the hierarchical structure for the registered group. We use the *idemix* library for the implementation purposes. Therefore, the protocols are based on the CL (Camenisch-Lysyanskaya) signature scheme. *idemix* is an anonymous credential system developed by IBM Research. As mentioned in Chapter 2 *idemix* is based on three main components, Issuer (group manager), Verifier (server), and the Prover (user). Group manager is responsible for issuing credentials to members of the group (users). Credentials are just like identities hold by users. The attributes credentials contain are signed digitally by the issuer and user can select any number of attributes to prove. After that, user can send the created proofs of possession. Created proofs can be verified by the server, this way server can conclude whether the attributes that the user claims are approved by the group manager. Since the proofs can only be opened by the group manager, the implementation also provides anonymity for users against the server. Moreover, users can authenticate themselves as valid members of the groups by just using the roles they hold as a group member, this is essential for the system, as in the proposed system Cloud should work in a need-to-know basis. Proposed system is implemented in Java since it aims to be a platform-free system to enhance mobility. Users can use devices varying from low-capacity mobile devices to PCs. An overview of the system can be found in Figure 3.1.

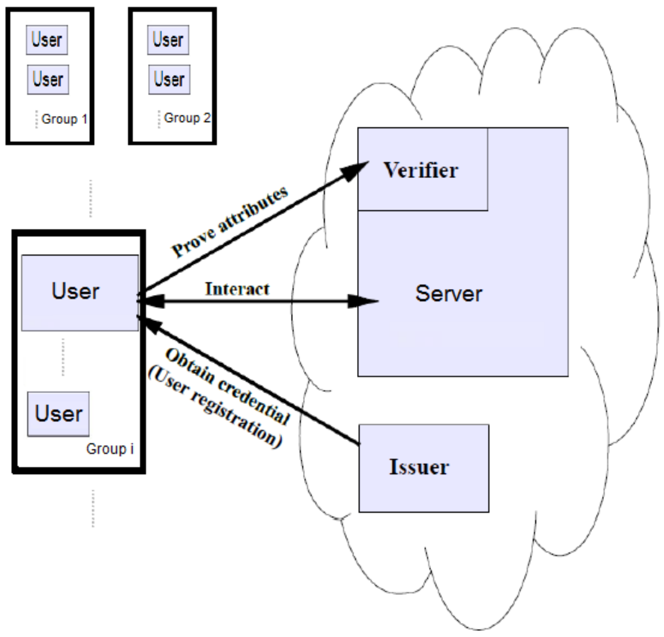


Figure 3.1: System Overview

An particular application of the proposed system can be summarized as follows. Suppose that, there are software development groups, which have any number of members in any number of roles within a customizable hierarchical structure. A group can consist of equals or it can have a strong hierarchy between the members. This hierarchy between the roles is defined in the group registration phase, and reflected to group access rights scheme. Every file may require different number of permissions for different kind of actions. A possible use case scenario of the proposed system is the following. Let there be a group consisting of 10 people. The group has five coders, two testers, two supervisors and a group manager. Only the group manager can be the Issuer. Hence, she interacts with each member to issue credentials which contain their roles and group identifiers and is capable of opening signatures, if necessary. After recieving a credential, whenever a user wants to interact with the server, she firstly authenticates herself to the verifier by sending her proof of possession of the attributes in the credential. After the proof is verified by the verifier, the user is authenticated. From that point on, user can interact with the Cloud Service, in this case a software development environment. Suppose that a user requests an action about a file from the server. The server firstly checks the required number of permissions for the requested action about that file and informs the online members about the request. Remember that, a coder may have one permission share, whereas the manager can have three. After a pre-determined time period, if the required number of permissions are collected, server grants the action. Otherwise, it informs the user about the failure. This idea of possessing different number of permission shares for different roles is mainly inspired by k out of l group signature schemes and allows us to support different kind of hierarchies, flat or strict, without interfering with their inner workings. 3.2

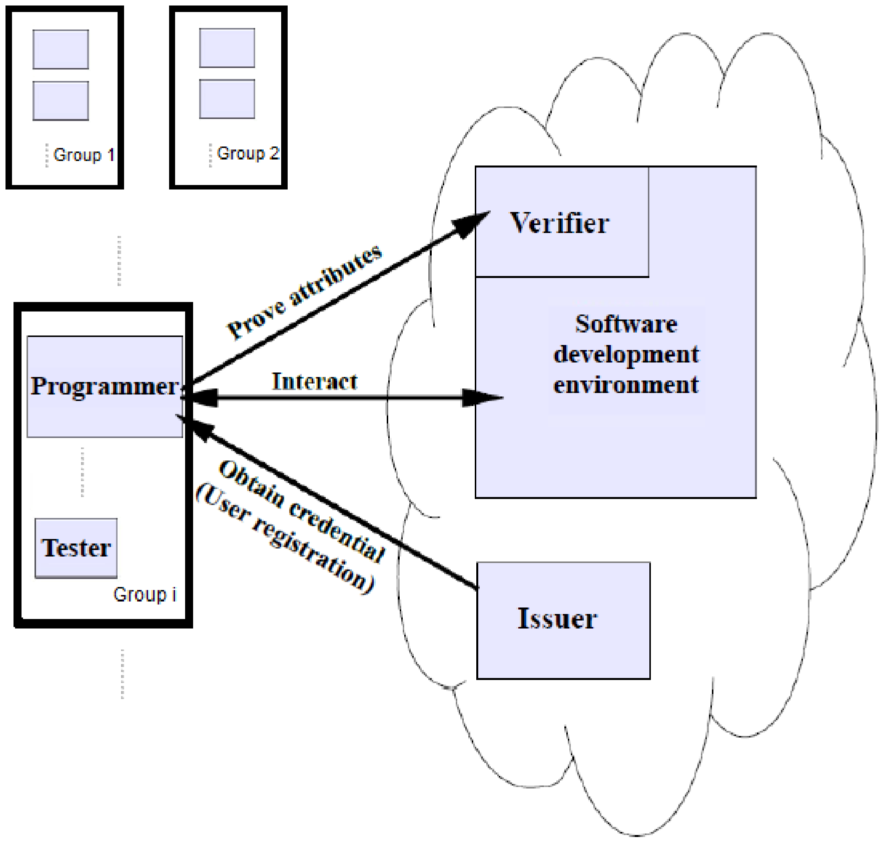


Figure 3.2: A specific application of the proposed system

3.1 Protocols

In this section, we provide the mathematical foundation of the CL signature scheme and present the protocols that we use throughout the implementation of the system.

3.1.1 Setup

The system parameters must be fixed and made public before any group registration. Group registration means generating group parameters using system parameters. The system parameters are given in the Table 3.1.

Table 3.1: The system parameters

ℓ_n	size of RSA modulus
ℓ_Γ	size of the commitment group modulus
ℓ_ρ	size of the prime order subgroup of Γ
ℓ_m	size of attributes
ℓ_{res}	number of reserved attributes in a certificate (data items and a digital signature by issuer on the data items)
ℓ_e	size of e values of certificates
ℓ'_e	size of the interval the e values are taken from
ℓ_v	size of the v values of the certificates
$\ell_\varnothing$	security parameter that governs the statistical zero-knowledge property
ℓ_k	security parameter
ℓ_H	domain of the hash function H used for the Fiat-Shamir heuristic
ℓ_r	security parameter required in the proof of security of the credential system
ℓ_{pt}	prime number generation returns composites with probability $1 - 1/2^{\ell_{pt}}$

Table 3.2: The symbol table

PU_{issuer}	public key of the issuer
PR_{issuer}	private key of the issuer
n_i	nonce with identifier i
attr	U defined in Issue Protocol Round 1
P_1	The proof generated in Issue Protocol Round 2
P_2	The proof generated in Issue Protocol Round 3
U	The cryptographic structure which contains attributes

3.1.2 Issuer Key Generation

Issuer key is used during issuance of credentials (user registration). The maximum number of attributes in a credential is determined by the length in the public key. ℓ being the number of attributes a key can contain, every credential can have $\ell - \ell_{res}$ number of attributes, where ℓ_{res} is reserved for the master secret.

Issuer generates a safe RSA key pair. For this, she generates safe primes p and q where $p = 2p'+1$ and $q = 2q'+1$. The issuer also generates parameters for CL signature scheme by choosing $S \in_R QR_n$, and $Z, R_1, .., R_l \in_R \langle S \rangle$. QR_n is group of quadratic residues mod n, $\langle S \rangle$ is the subgroup generated by S. The order of S must be $\#QR_n = p'q'$. Furthermore, issuer chooses $x_X, x_{R_1}, ..., x_{R_l} \in_R [2, p'q' - 1]$ and computes $Z = S^{xz}$, $R_i = S^x_R$ for $1 \leq i \leq l$.

The issuer's public key is $PU_{issuer} = (n, S, Z, R_1, ..., R_l, P)$, and the private key is $PR_{issuer} = (p, q)$.

3.1.3 Issuance Protocol (User Registration)

The issuance protocol is performed between the user (in *idemix* terms, user is called recipient) and the group manager (issuer) interactively. This can be seen as a new user registration phase of the proposed system. After issuance of a credential a group member becomes able to use the features of the Cloud service. In that sense, the member is already registered as a group member, however as a result of this interaction, the user gets a credential that is issued by the issuer, and hence she is registered as a valid user of the Cloud service. In the proposed system, the issuer is in the Cloud but it is a different entity than the verifier. We propose to differentiate between these two entities, since the group manager is also responsible for member revocation and opening signatures if necessary. In this way, we aim to keep group management issues of this kind outside of the main system. For obtaining a new credential, the user needs a credential structure definition, which can differ from group to group. A credential structure definition defines the attribute structure of the credential to be issued. In our case, credentials contain the roles in the group (e.g. coder, supervisor, manager, etc.). This can be extended, however, and credentials can contain more than one attribute in order to enhance the access right scheme. Both the recipient and the issuer must have the same definition in order to perform the issue protocol. The protocol flow can be seen in Figure 3.3, and the symbols can be seen in Table 3.2

The protocol is as follows:

- User (recipient) initiates the protocol by sending a request to the group manager (issuer).
- Issuer initializes group and system parameters. Then, issuer computes a nonce (n_1) with respect to the initialization and sends it to recipient. For details, see Algorithm 1.
- Recipient receives the nonce (n_1), which she uses to compute a cryptographic message with the properties of the desired credential, and sends it to issuer using Algorithm 2.
- Issuer receives the message sent in round 2, uses it to create the cryptographic part of the credential (by signing it with the master key), and sends it to recipient as described in Algorithm 3.
- Recipient receives the credential, and saves it by following the steps in Algorithm 4.

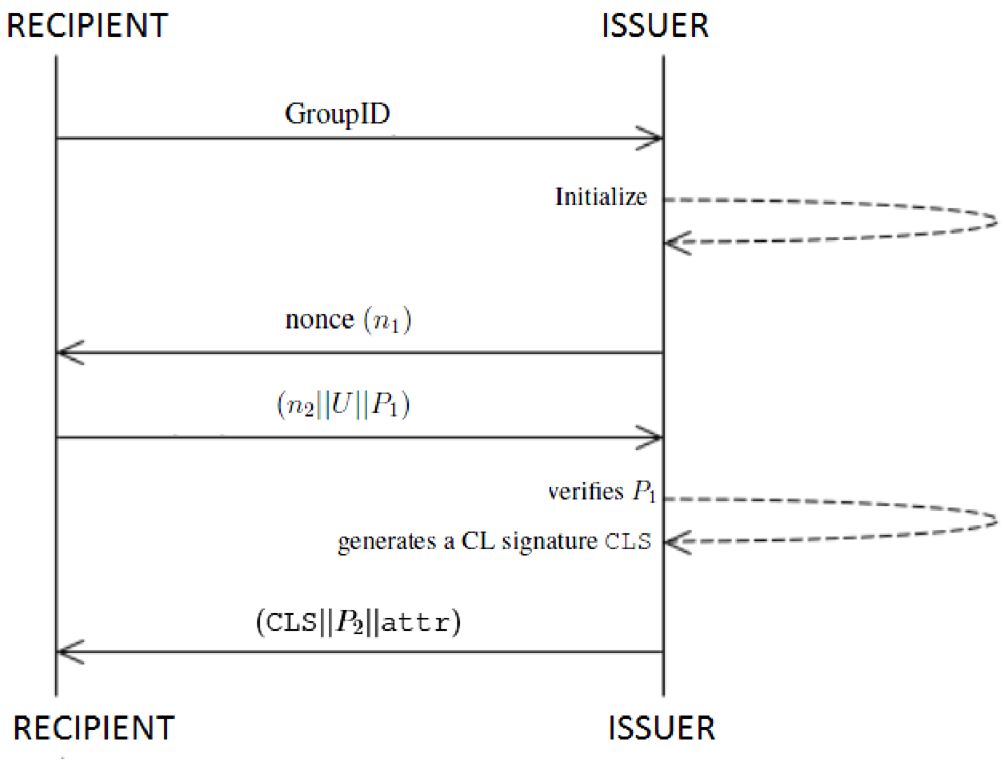


Figure 3.3: Overview of the Issue protocol

Algorithm 1 Issue Protocol - Round 0

```
ISSUER chooses a random nonce $n_1 \in_R \{0,1\}^{\ell_\varnothing}$
Load attribute structures from $S$ to $A$ (both ISSUER and RECIPIENT)
ISSUER $\rightarrow$ RECIPIENT : $n_1$
```

Algorithm 2 Issue Protocol - Round 1

```
RECIPIENT chooses a random integer $v' \in_R \pm\{0,1\}^{\ell_n+\ell_\varnothing}$
RECIPIENT computes: $U := S^{v'} \prod_{j \in A} R_j^{m_j} \mod n$, for $j \in A$
computes a non-interactive proof to verify $U$ is computed correctly.
    $U \equiv \pm S^{v'} \prod_{k \in A} R_k^{m_k} \mod n$
Choose $\tilde{m}_j \in_R \pm\{0,1\}^{\ell_m+\ell_\varnothing+\ell_H+1}$
RECIPIENT computes: $\tilde{U} := S^{\tilde{v}'} \prod_{j \in A} R_j^{\tilde{m}_j} \mod n$
RECIPIENT computes: $c := H(context||U||\tilde{U}||n_1)$
RECIPIENT responses to the challenge:
    $\hat{v}' := \tilde{v}' + cv'$
    $s_A := (\hat{m}_j := \tilde{m}_j + cm_j)_{j \in A}$
    $P_1 := (c, \hat{v}', s_A)$
RECIPIENT chooses $n_2 \in_R \{0,1\}^{\ell_\varnothing}$
RECIPIENT $\rightarrow$ ISSUER : $U, P_1, n_2$
RECIPIENT stores $A_k$, $v'$, $\texttt{context}$.
```

3.1.4 Authentication Protocol

Whenever a user is connected to the system, she must authenticate herself in order to begin her interaction with the Cloud as a registered user. For that, she has to have a credential,

Algorithm 3 Issue Protocol - Round 2

```
ISSUER verifies $P_1$:
    $\hat{U} := U^{-c}(S^{\hat{v}'}) \prod_{j \in A} R_j^{\hat{m}_j} \mod n$
    $\hat{c} := H(context||U||\hat{U}||n_1)$
if $\hat{c} \neq c$ then
    verification fails.
end if
ISSUER checks
    $\hat{v}' \in \pm\{0,1\}^{\ell_n + 2\ell_\varnothing + \ell_H + 1}$
    $\hat{m}'_i \in \pm\{0,1\}^{\ell_m + \ell_\varnothing + \ell_H + 2}$, for all, $i \in A$
if length check fails then
    verification fails.
end if
ISSUER generates a CL Signature on the attributes:
    Choose a random prime : $e \in_R [2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$
    Choose a random integer $\tilde{v} \in_R \{0,1\}^{\ell_v - 1}$, and compute $v'' := 2^{l_v - 1} + \tilde{v}$
    Compute: $Q := \frac{Z}{US^{v''} \prod_{i \in A} R_i^{m_i}} \mod n$ and $A := Q^{e^{-1} \mod p'q'} \mod n$
    ISSUER stores: $Q, A_k, v'', \texttt{context}$
ISSUER creates the following proof of correctness:
$SPK\{(e^{-1}) := A \equiv \pm Q^{e^{-1}} \mod n\}(n_2)$:
    Compute $\tilde{A} := Q^r(\mod n)$, for $r \in_R Z^*_{p'q'}$
    Compute $c' := H(\texttt{context}||Q||A||\tilde{A}||n_2)$
    Compute $s_e := r - c'e^{-1}(\mod p'q')$
    $P_2 := (s_e, c')$
ISSUER → RECIPIENT: $(A, e, v''), P_2, (m_i)_{i \in A}$
```

Algorithm 4 Issue Protocol - Round 3

```
RECIPIENT computes: $v := v'' + v'$
RECIPIENT verifies $(A, e, v)$ using CL-signature verification:
    Checks $e$ is a prime and $e \in [2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}]$
    Computes $Q := \frac{Z}{S^v \prod_{i \in S} R_i^{m_i}} \mod n$
    Computes $\hat{Q} := A^e \mod n$
if $Q \not\equiv \hat{Q} \mod n$ then
    abort
end if
RECIPIENT verifies $P_2$
    Computes $\hat{A} := A^{c' + s_e e} S^{v' s_e} \mod n$
    Computes $\hat{c} = H(\texttt{context}||Q||A||\hat{A}||n_2)$
if $\hat{c} \neq c'$ then
    abort
end if
Store the credential: $(m_i)_{i \in A}, (A, e, v)$
```

containing her group ID and role. Any messages before a user has been authenticated is discarded by the server and user receives a notification to properly authenticate herself. The protocol can be summarized as follows. A user with a valid credential creates a proof using that credential (buildProof). Then, she sends it to verifier (server) and if the verifier decides the proof is valid (verifyProof), the user becomes authenticated. The user needs a proof specification to specify which attributes she likes to prove to the verifier. The verifier also needs the proof specification to know which attributes the proof proves. Therefore, they need to share the proof specification beforehand. The overview of the Authentication Protocol can be found in the Figure 3.4.

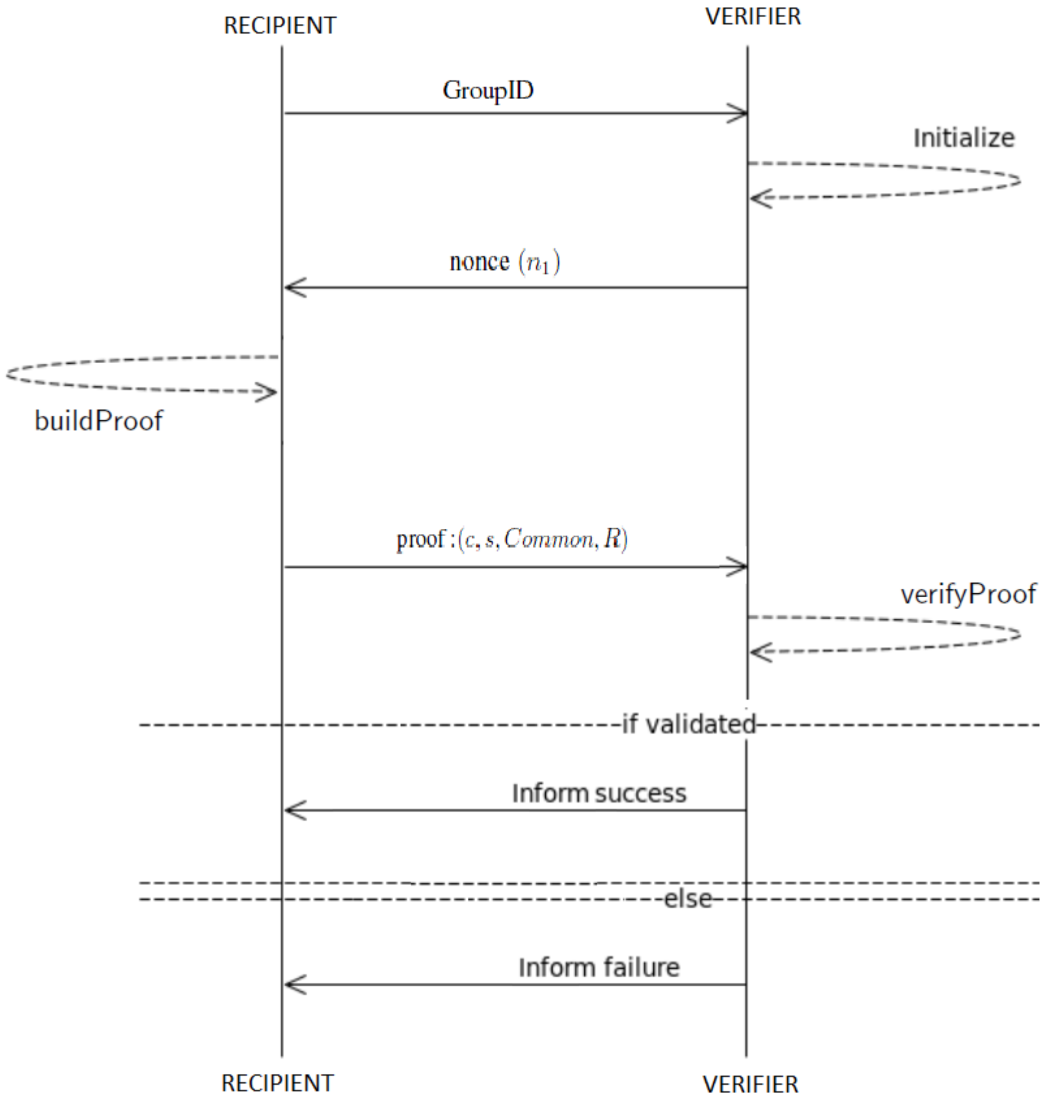


Figure 3.4: Authentication Protocol

The subprotocols are explained in the next sections.

3.1.4.1 Building a Proof

The **input** is $m_1, \{cred\}, S, n_1$
$\{cred\}$ is the credential. The nonce n_1 is provided by the VERIFIER and sent to the

PROVER. Let $V = \{v_1, ..., v_t\}$ be the set of values in the credentials held by the prover. In the setup phase, the PROVER generates $\hat{v}_i \in_R \{0,1\}^{\ell_m+\ell_\varnothing+\ell_H}$ for each hidden identifier in V. Then, it computes t-values:

$$\begin{aligned} SPK\{(e, \{m_i : i \in A_h\}, v) : \quad & \frac{Z}{\prod_{i \notin A_h} R_i^{m_i}} \equiv \pm A^e S^v \prod_{i \in A_h} R_i^{m_i} \pmod n \\ & \wedge\, m_i \in \{0,1\}^{\ell_m+\ell_\varnothing+\ell_H+2} \quad \forall i \in A_h \\ & \wedge\, e - 2^{\ell_e-1} \in \{0,1\}^{\ell'_e+\ell_\varnothing+\ell_h+2}\} \ (n_1) \end{aligned}$$

After that, the PROVER chooses $r_A \in_R \{0,1\}^{\ell_n+\ell_\varnothing}$ and computes the randomized CL signature (A', e, v'), where

$$\begin{aligned} A' &:= AS^{r_A} \pmod n, \\ v' &:= v - er_A \text{ (in } \mathbb{Z}). \end{aligned}$$

It additionally computes $e' := e - 2^{\ell_e-1}$. To compute t-values, PROVER chooses random integers

$$\begin{aligned} \tilde{e} &\in_R \pm\{0,1\}^{\ell'_e+\ell_\varnothing+\ell_H}, \\ \tilde{v}' &\in_R \pm\{0,1\}^{\ell_v+\ell_\varnothing+\ell_H}. \end{aligned}$$

For each identifier $i \in I$, it recovers the corresponding random value $\tilde{m}_i$, computed in step 0.1 of buildProof. Then it computes

$$\tilde{Z} := (A')^{\tilde{e}} \Big(\prod_{i \in I} R_i^{\tilde{m}_i} \Big) (S^{\tilde{v}'}) \mod n.$$

The **output** is t-value $\tilde{Z}$ and common value A'.
Next, the PROVER computes the challenge: $c := H(context, Common, T, n_1)$. To compute the responses, it computes the following s-values in $\mathbb{Z}$.

$$\begin{aligned} \hat{e} &:= \tilde{e} + ce' \ (= \tilde{e} + c(e - 2^{\ell_e-1})), \\ \hat{v}' &:= \tilde{v}' + cv', \\ \hat{m}_i &:= \tilde{m}_i + cm_i for i \notin A_r. \end{aligned}$$

The **output** is the proof :$(c, s, Common, R)$.

3.1.4.2 Verifying a Proof

The **input** is S, P = $(c, s, Common), n_1$.
To compute $\hat{t} - values$ [1], the VERIFIER computes

$$\hat{T} := \left(\frac{Z}{(\prod_{i \in A_r} R_i^{m_i} (A')^{2^{\ell_e - 1}}}\right)^{-c} (A')^{\hat{e}} \left(\prod_{i \in A_{\bar{r}}} R_i^{\hat{m_i}}\right) S^{\hat{v}'} \mod n$$

Then it verifies the lengths:

$$\begin{aligned} \hat{m_i} &\in \pm\{0,1\}^{\ell_m + \ell_\varnothing + \ell_H + 1}, for i \in A_{\bar{r}} \\ \hat{e} &\in \pm\{0,1\}^{\ell_m + \ell_\varnothing + \ell_H + 1} \end{aligned}$$

If the length checks do not fail, VERIFIER computes the verification challange:

$$c := H(context, Common, \hat{T}, n_1)$$

After that, it verifies the equality of challenge and if $c \equiv \hat{c}$, it accepts the proof P.

3.1.5 File Access Protocol

Once a user is authenticated, the server knows which group the user belongs to, and the root directory for the user is already initialized. From that point on, the user can request directory listing or an action on a file (read, write, execute, compile). Group managers can define which access requires how many permissions in the group registration phase. For example, a readme file can be read without requiring any other permissions than the request, whereas a classified document may need five permissions in order to be compiled.

As mentioned in the Section 3, users may have different number of shares about files according to their roles and the access right scheme. Groups can have a different inner hierarchies, and they are free to express it in their access right schemes during registration. After a file is requested, server informs the other members of the group, and waits for permissions until the pre-determined time is over or required number of permissions are collected. After that, server responds to the request. The file access protocol takes place between all online members of the group and the server and can be summarized as follows:

- **Round 0** An authenticated user ($u_k \in G_a$) requests an action over a file F.
- **Round 1** Server informs online members ($u_i \in G_a, i \neq k$) about the request.

[1]The values of the form $t = g^r$ used in the first flow of the protocol will be called "t-value", while the responses computed in the third flow of the form $s = r - c^\alpha$ will be reffered to as "s-values".

- **Round 2** ($u_i \in G_a, i \neq k$) respond with permission, or not.
- **Round 3** After required number of permissions are collected, or the time threshold is exceeded, server responds to u_k.

The sequence diagram for the file access protocol can be seen in Figure 3.5:

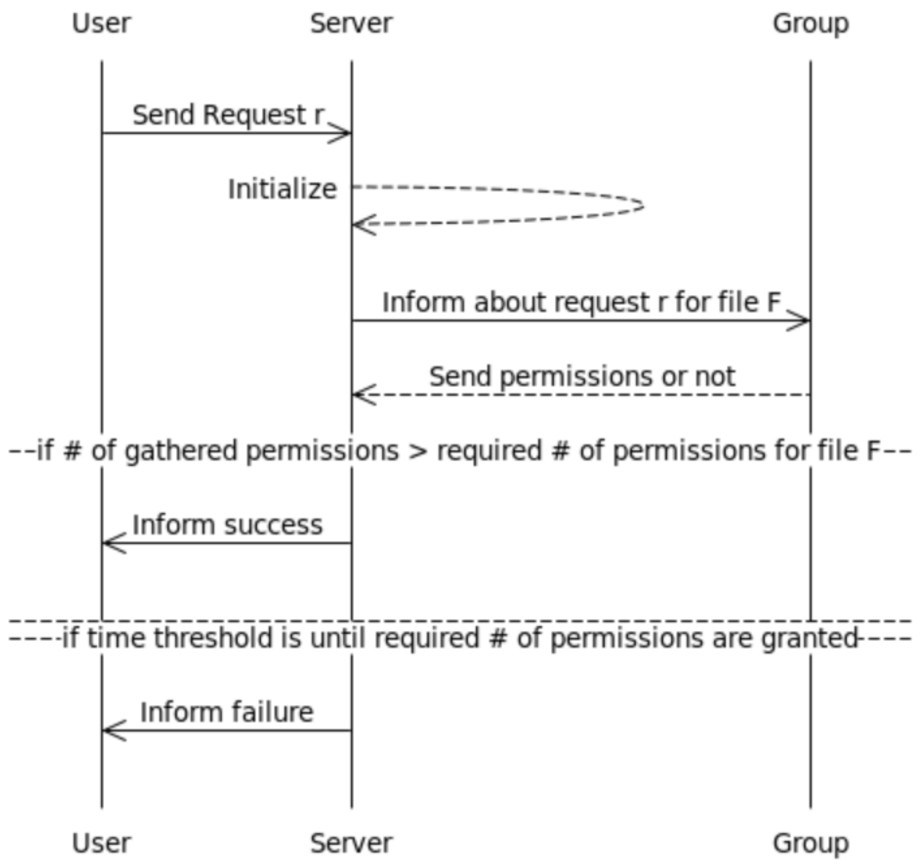

User
Server
Group
Send Request r
Initialize
Inform about request r for file F
Send permissions or not
--if # of gathered permissions > required # of permissions for file F--
Inform success
----if time threshold is until required # of permissions are granted----
Inform failure
User
Server
Group

Figure 3.5: File Access Protocol

3.1.6 User Revocation Process

We also develop a user revocation process. Revocation works as explained below Issuer basically changes the list of valid credentials, which is a part of her public key, and shares it in the Group paramaters, which are public. Since the member to be revoked cannot update her credential, her existing credential can no more produce valid proofs. As a result, the owner of the credential becomes revoked. The non-revoked members can update their credentials, based on the new public key, following the update protocol as described below and the details can be found in Algorithm **??**.

Firstly, Issuer loads $A_k, (m_i)_{A_k}, Q$ and v'' from the non-revoced members credential, and the generates a CL signature on the attributes. For that she computes a random integer

Algorithm 5 Credential Update

```
ISSUER loads $A_k, (m_i)_{A_k}, Q$ and $v''$ from the credential, it wants to update
ISSUER reads previously saved elements from update file.
ISSUER generates a CL signature on the attributes.
    Chooses a random prime : $e \in_R [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}]$.
    Chooses a random integer: $\tilde{v} \in_R \{0,1\}^{l_v-1}$
ISSUER computes:
    $\bar{v}'' := 2^{l_v-1} + \tilde{v}$ and $\Delta v'' := \bar{v}'' - v''$.
ISSUER computes:
    $\bar{Q} := \frac{Q}{(\prod_{i \in A_{KN}} R_i^{\Delta m_i}) S^{\Delta v''}} \mod n$ and $\bar{A} := \bar{Q}^{e^{-1} \bmod p'q'} \mod n.$,
    where $(\bar{m}_i)_{A_k}$ are the updated values, and $\Delta m_i = \bar{m}_i - m_i$
$Q := \bar{Q},\ v'' := \bar{v}'',\ m_i := \bar{m}_i$ and $A := \bar{A}$.
ISSUER creates the proof of correctness: $P_2 := SPK\{(e^{-1}) : A \equiv \pm Q^{e^{-1}} \mod n\}(n_2)$
ISSUER updates the following elements in file (for use in credential update).
    $Q$
    $v''$
    $\{m_i : i \in A_k\}$
--------------------------------------------------------------------------
ISSUER sends $(A, e, v''), P_2$, and $(m_i)_{i \in A_k}$ to the RECIPIENT.
--------------------------------------------------------------------------
RECIPIENT verifies $P_2$
    Compute $Q$ and $\hat{Q} = A^{c'} Q^{s_e}$.
    Compute $\hat{c} = H(\texttt{context}||Q||A||\hat{Q}||n_2)$
if $\hat{c} \neq c'$ then
    abort
end if
RECIPIENT computes and stores $v = v' + v''$
RECIPIENT verifies $(A, e, v)$
if $e$ is prime and $e \in [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}]$ then
    abort
end if
if $Z \equiv A^e R_1^{m_1} ... R_l^{m_l} S^v \mod n$ then
    abort
end if
output: credential $(m_1, ..., m_l, (A, e, v))$
```

$\tilde{v}$ and a random prime e:

$$e \in_R [2^{l_e-1}, 2^{l_e-1} + 2^{l'_e-1}]$$

$$\tilde{v} \in_R \{0,1\}^{l_v-1}$$

Then she computes:

$$\bar{Q} := \frac{Q}{(\prod_{i \in A_{KN}} R_i^{\Delta m_i}) S^{\Delta v''}} \mod n \text{ and } \bar{A} := \bar{Q}^{e^{-1} \bmod p'q'} \mod n$$

Then she creates the proof of correctness:

$$P_2 := SPK\{(e^{-1}) : A \equiv \pm Q^{e^{-1}} \mod n\}(n_2)$$

where $(\bar{m_i})_{A_k}$ are the updated values, and $\Delta m_i = \bar{m_i} - m_i$, $Q := \bar{Q}$, $v'' := \bar{v}''$, $m_i := \bar{m_i}$, $A := \bar{A}$. Then she sends the CL Signature, P_2 and updated attributes ($m_i \in A$) to recipient.

After recipient recieves the message, firstly she verifies P_2, by computing Q ,$\hat{Q}$ and the challenge, where:

$$\hat{Q} = A^{c'} Q^{s_e}$$

$$\hat{c} = H(\texttt{context}||Q||A||\hat{Q}||n_2)$$

Then recipient verifies the CL signature and if the signature is correct, similar to Algorithm 4, saves the updated credential $(m_1, ..., m_l, (A, e, v))$.

Chapter 4

Implementation Details

In this chapter we provide a detailed explanation of the implementation. Firstly, we give information about how we achieved the adoption of the *idemix* library to implement the procedures in the CL signature scheme. Later, we discuss how the client-server architecture is implemented and details about the message structures that are used in the protocols.

We develop the proposed system in Java, in order to achieve independency from operating systems and platforms. The implementation has been done in Intel Core i7-2670QM quad-core (2.20GHz / 3.10GHz3 with Turbo Boost4) with 8GB (4GB x2) DDR3-SDRAM-1333, with the operating system Windows 7 Professional 64 bit. Eclipse has been used as the developing environment.

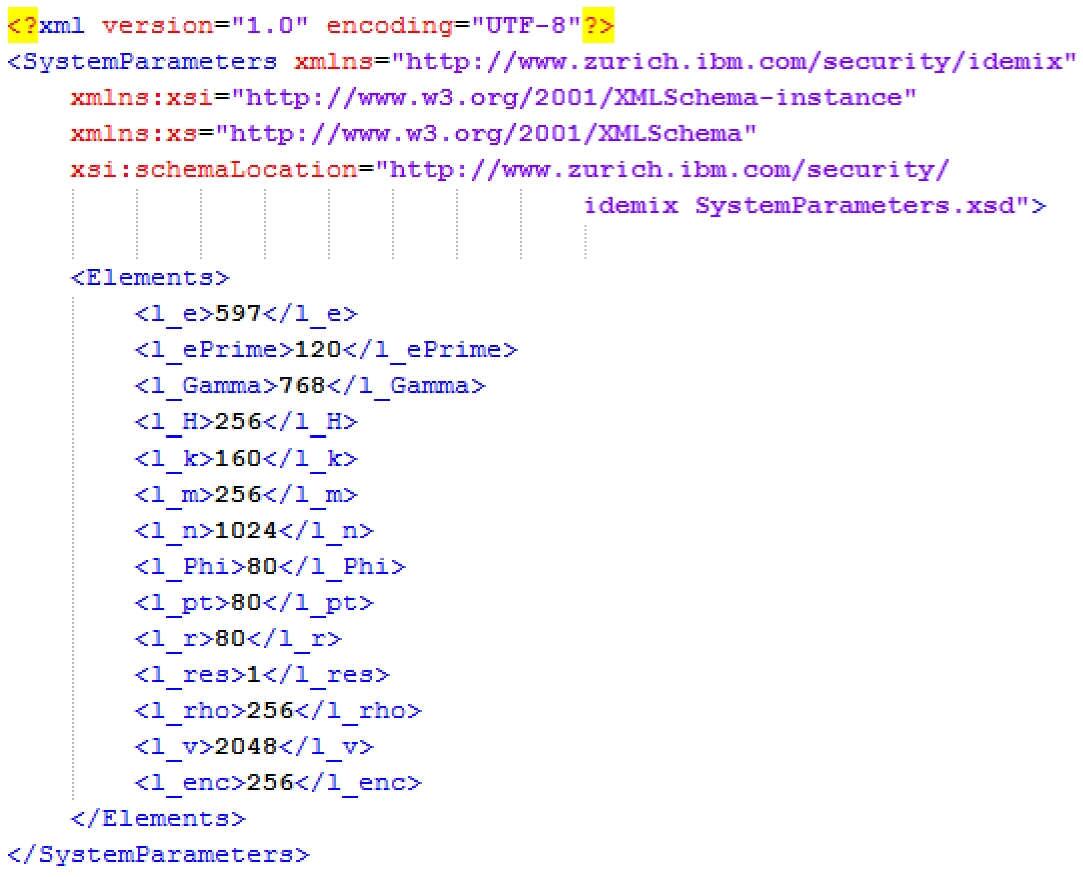
4.1 Adoption of *idemix* Library

We used the *idemix* library for implementing the protocols, namely member addition / revocation and authenticaton. However, since we want an interactive environment, we have to modularize the functions and make them suitable for a client-server architecture. In this section, we present how we achieved the adoption of *idemix* library and show some examples of used structures.

4.1.1 System Setup

The *idemix* library requires having general parameters to be used in the system, which are seperated into two:

- **System Parameters**: These are public parameters containing bit lenghts and probabilities that the generated primes are truly primes, and are used to generate group parameters during group registration. An example of a system parameters file can be seen in Figure 4.1. The related list of the parameters can be found in Table 3.1



```
<?xml version="1.0" encoding="UTF-8"?>
<SystemParameters xmlns="http://www.zurich.ibm.com/security/idemix"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xsi:schemaLocation="http://www.zurich.ibm.com/security/
                                idemix SystemParameters.xsd">

    <Elements>
        <l_e>597</l_e>
        <l_ePrime>120</l_ePrime>
        <l_Gamma>768</l_Gamma>
        <l_H>256</l_H>
        <l_k>160</l_k>
        <l_m>256</l_m>
        <l_n>1024</l_n>
        <l_Phi>80</l_Phi>
        <l_pt>80</l_pt>
        <l_r>80</l_r>
        <l_res>1</l_res>
        <l_rho>256</l_rho>
        <l_v>2048</l_v>
        <l_enc>256</l_enc>
    </Elements>
</SystemParameters>
```

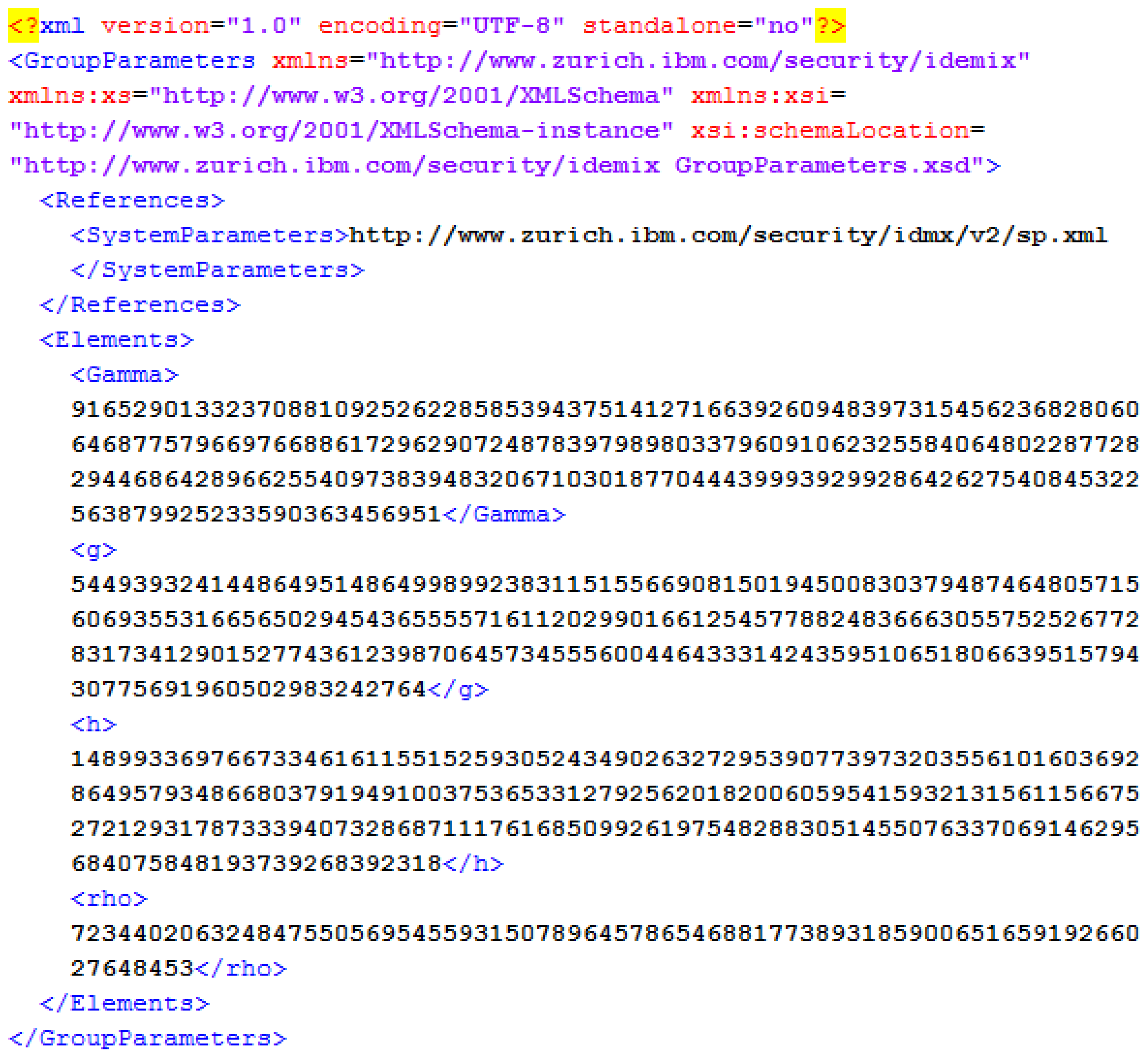
Figure 4.1: System parameters file

- **Group Parameters** Group parameters are generated from system parameters and used to generate the Issuer Key and the Master Key for further interactions, using CL signature scheme. An example can be seen in Figure 4.2.

During group registration, the group manager also indicates the roles and the corresponding number of permission shares for the roles, which defines the group hierarchy for further interactions.

4.1.2 Implementation of the protocols

The *idemix* library contains test cases to show how the library can be used for authentication purposes. However, the test cases do not contain any implementations which can be used directly in a Client - Server architecture. The main challange is to modularize the code so that the building blocks for protocols work properly and do not use unnecessary capabilites. We apply the same process to user registration, group registration and most importantly, the authentication (signing and verification). The generation of group parameters and the issuance protocol are quite the same regardless of the complexity of

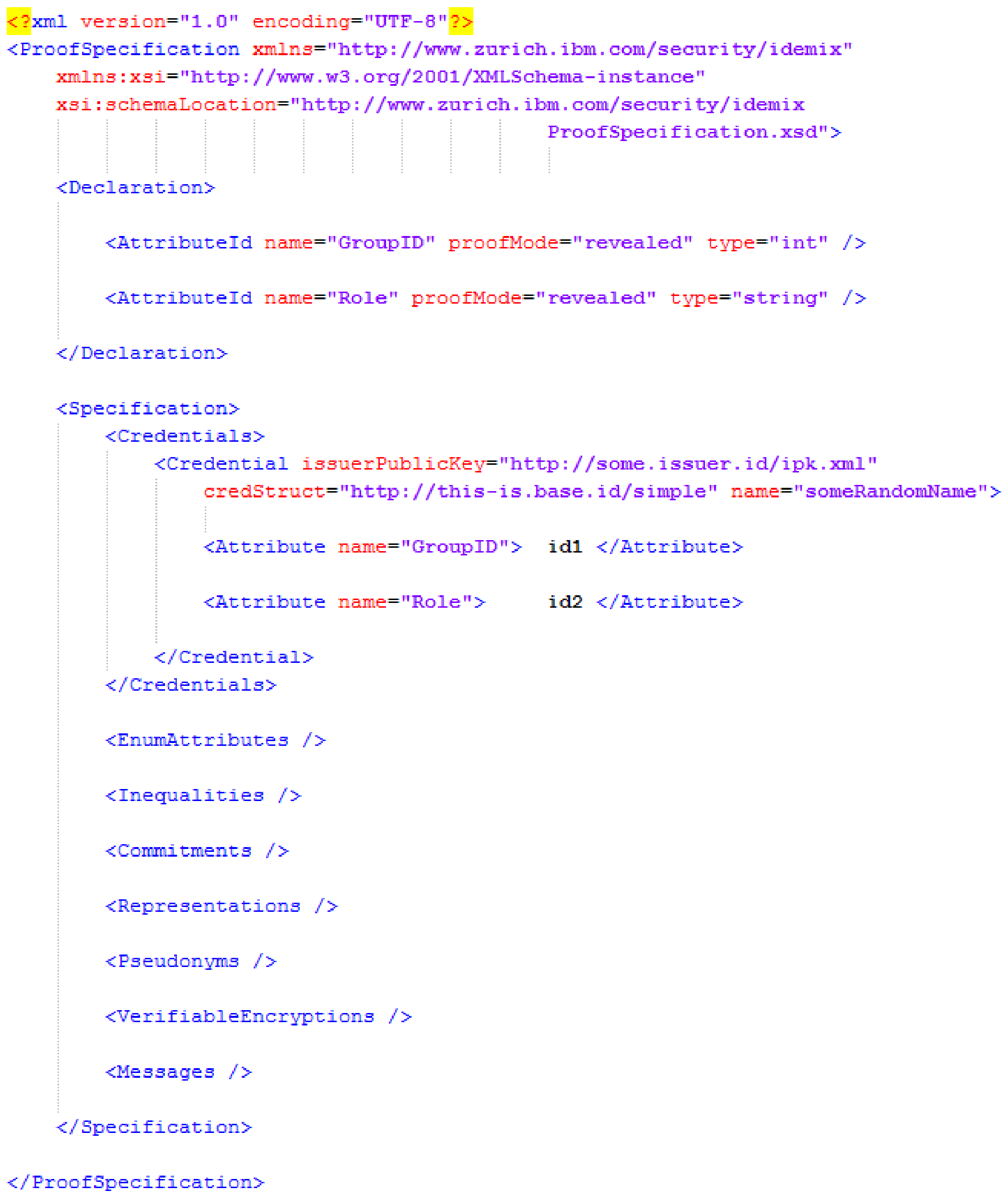


```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<GroupParameters xmlns="http://www.zurich.ibm.com/security/idemix"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://www.zurich.ibm.com/security/idemix GroupParameters.xsd">
  <References>
    <SystemParameters>http://www.zurich.ibm.com/security/idmx/v2/sp.xml
    </SystemParameters>
  </References>
  <Elements>
    <Gamma>
    91652901332370881092526228585394375141271663926094839731545623682806O
    646877579669766886172962907248783979898033796091062325584064802287728
    294468642896625540973839483206710301877044439993929928642627540845322
    56387992523359036345695l</Gamma>
    <g>
    544939324144864951486499899238311515566908150194500830379487464805715
    606935531665650294543655557161120299016612545778824836663055752526772
    831734129015277436123987064573455560044643331424359510651806639515794
    30775691960502983242764</g>
    <h>
    148993369766733461611551525930524349026327295390773973203556101603692
    864957934866803791949100375365331279256201820060595415932131561156675
    272129317873339407328687111761685099261975482883051455076337069146295
    684075848193739268392318</h>
    <rho>
    723440206324847550569545593150789645786546881773893185900651659192660
    27648453</rho>
  </Elements>
</GroupParameters>
```

Figure 4.2: Group parameters file

the credentials or proofs. The main focus in adopting the library is in the authentication process. We modularize the functions of the *idemix* library into smaller functions as we describe in the protocols. The details of signing and verifying functions are as follows:

- **Client Side**:
  - **beginSigning** : This function is used in the client side and starts the signing process. Firstly, it initializes the system parameters and recieves the credential and the proofSpec from the interface, and then, returns a HashMap containing attributes. An example of a proof specification file of *idemix* can be found in Figure 4.3.
  - **finalizeSigning**: This function takes the nonce (see Figure 4.5), the *hashMap* of attributes, the proofSpec and the master key as parameters. Then, it creates

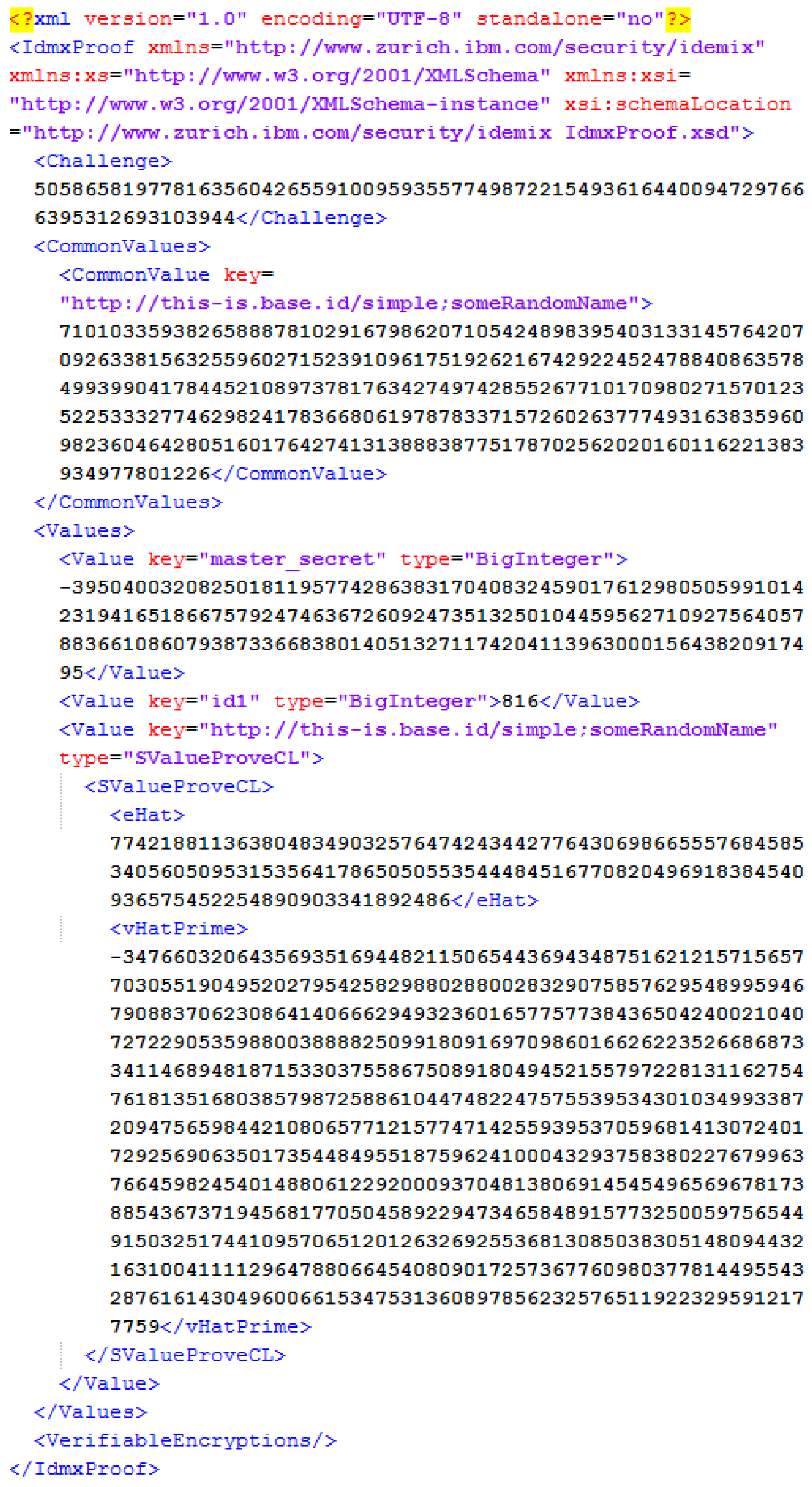


```xml
<?xml version="1.0" encoding="UTF-8"?>
<ProofSpecification xmlns="http://www.zurich.ibm.com/security/idemix"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.zurich.ibm.com/security/idemix
                                        ProofSpecification.xsd">

    <Declaration>

        <AttributeId name="GroupID" proofMode="revealed" type="int" />

        <AttributeId name="Role" proofMode="revealed" type="string" />

    </Declaration>

    <Specification>
        <Credentials>
            <Credential issuerPublicKey="http://some.issuer.id/ipk.xml"
                credStruct="http://this-is.base.id/simple" name="someRandomName">

                <Attribute name="GroupID">  id1 </Attribute>

                <Attribute name="Role">     id2 </Attribute>

            </Credential>
        </Credentials>

        <EnumAttributes />

        <Inequalities />

        <Commitments />

        <Representations />

        <Pseudonyms />

        <VerifiableEncryptions />

        <Messages />

    </Specification>

</ProofSpecification>
```

Figure 4.3: A sample proof specification structure

a new prover using these parameters, and builds and returns proof p (see Figure 4.4), which is a cryptographic file containing the attributes issued before.

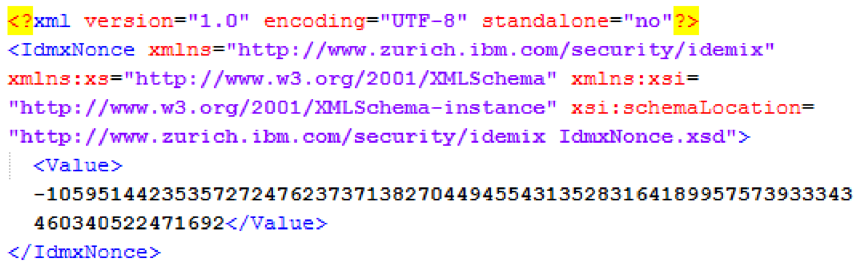
- **Server Side**:
  - **sendNonce**: After receiving the request and a group identification number, verifier initializes the system parameters, generates a nonce, stores it as a tuple with thread number and sends the nonce file back to client.
  - **verifySignature**: This function takes a proof file, the related nonce and proofSpec as parameters and returns the set of revealed values. Using the revealed



```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<IdmxProof xmlns="http://www.zurich.ibm.com/security/idemix"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation
="http://www.zurich.ibm.com/security/idemix IdmxProof.xsd">
  <Challenge>
  5058658197781635604265591009593557749872215493616440094729766
  6395312693103944</Challenge>
  <CommonValues>
    <CommonValue key=
    "http://this-is.base.id/simple;someRandomName">
    71010335938265888781029167986207105424898395403133145764207
    09263381563255960271523910961751926216742922452478840863578
    49939904178445210897378176342749742855267710170980271570123
    52253332774629824178366806197878337157260263777493163835960
    982360464280516017642741313888838775178702562020160116221383
    934977801226</CommonValue>
  </CommonValues>
  <Values>
    <Value key="master_secret" type="BigInteger">
    -3950400320825018119577428638317040832459017612980505991014
    23194165186675792474636726092473513250104459562710927564057
    88366108607938733668380140513271174204113963000156438209174
    95</Value>
    <Value key="id1" type="BigInteger">816</Value>
    <Value key="http://this-is.base.id/simple;someRandomName"
    type="SValueProveCL">
      <SValueProveCL>
        <eHat>
        7742188113638048349032576474243442776430698665557684585
        3405605095315356417865050553544484516770820496918384540
        936575452254890903341892486</eHat>
        <vHatPrime>
        -347660320643569351694482115065443694348751621215715657
        7030551904952027954258298802880028329075857629548995946
        7908837062308641406662949323601657757738436504240021040
        7272290535988003888825099180916970986016626223526686873
        3411468948187153303755867508918049452155797228131162754
        7618135168038579872588610447482247575539534301034993387
        2094756598442108065771215774714255939537059681413072401
        7292569063501735448495518759624100043293758380227679963
        7664598245401488061229200093704813806914545496569678173
        8854367371945681770504589229473465848915773250059756544
        9150325174410957065120126326925536813085038305148094432
        1631004111129647880664540809017257367760980377814495543
        2876161430496006615347531360897856232576511922329591217
        7759</vHatPrime>
      </SValueProveCL>
    </Value>
  </Values>
  <VerifiableEncryptions/>
</IdmxProof>
```

Figure 4.4: A sample Proof file

values, it confirms the group identification number and determines the role of the connected client.



```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<IdmxNonce xmlns="http://www.zurich.ibm.com/security/idemix"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"http://www.zurich.ibm.com/security/idemix IdmxNonce.xsd">
  <Value>
  -105951442353572724762373713827044945543135283164189957573933343
  460340522471692</Value>
</IdmxNonce>
```

Figure 4.5: An example of a nonce file

4.2 Client - Server Architechture

We need to use socket programming in order to connect the clients to the server. We use the OCSF (The Object Client-Server Framework) provided in [30]. The client side of OCSF consists of an abstract class called AbstractClient. This class provides the facilities needed to connect to and exchange objects with servers. The method handleMessageFromServer is modified so that the client takes appropiate action for the received message. The server side consists of two classes, namely AbstractServer and ConnectionToClient. AbstractServer is responsible for listening to incoming connections and the threads for each connection, whereas ConnectionToClient handles the connections to clients.

We implement handleMessageFromServer functions so that both sides can process the protocols we use properly. We also apply a few modifications to the existing ConnectionToClient scheme. Every connection now holds a ClientProperties file, which keeps record of the current ProofSpec, the current nonce, the current proof of authentication and the group identification number. Once a client is connected, during the first step of the authentication, she informs the server about her group id. From that point on, the verifier knows which ProofSpec to use and keeps it in the current ProofSpec. The current nonce holds the last sent nonce which is used during the third step of authentication. The current Proof keeps the information of last sent proof. This is checked periodically before any interaction since in case of user revocation, server can confirm whether the proved attributes of a credential are still valid or not.

Once the server starts to listen, whenever a new connection arrives, the server starts a thread. Before authentication, no messages from that thread are accepted, and all af them are replied with a message informing the user that she needs to authenticate herself. If a message with proper header arrives, the authentication protocol begins. After a successful authentication, the client starts to interact with the server, requesting actions for files and recieving permission requests from the server.

4.3 Messages

Since we use the OCSF for client server architechture, we need to define a message structure, which can be easily upcasted to class Object and downcasted from it. Object is the root of the class hierarchy in Java. Every class has Object as a superclass. So we implement our own MessageStruct, which consists of a header, a file and a string, called ”details”. We use the header part to distinguish between messages. File part keeps the cryptographic file (nonce, proof etc). ”details” differ from message to message and hold necessary information in order to progress through protocols.

Chapter 5

Performance Evaluation

To evaluate the performance of the system, we perform a simulation based analysis so that we see how the system behaves when the number of connected clients increases. In this section, we provide information about the simulation and a discussion about its results. We firstly give the unit times for the main operations, and then present our simulation model. Lastly, we discuss the results via performance graphs.

All unit times are measured in Intel Core i7-2670QM quad-core (2.20GHz / 3.10GHz3 with Turbo Boost4) with 8GB (4GB x2) DDR3-SDRAM-1333, with the operating system Windows 7 Professional 64 bit. Every measurement, including the stress tests and unit time measurements have been done 50 times and the average values are reported. For the simulation based analyse, simulation time has been taken as 360 minutes.

5.1 Unit Operations

There are three main unit operations in the protocols. In the Issue Protocol, the unit operation is issuing a credential. In the Authentication Protocol, creating the proof, and validating the sent proof are unit operations. We consider three types of proofs, 'simple', 'medium', and 'complex', Table 5.1 shows the revealed and unrevealed number of attributes in each proof. The measured unit times can be found in the table 5.2.

Table 5.1: The number of attributes in proofs

	Revealed Attributes	Unrevealed Attributes	Total Attributes
Simple Proof	2	0	2
Medium Proof	4	0	4
Complex Proof	16	16	32

Group registration can also be considered as a unit operation. In order to register a group, we have to fix the public system parameters, as mentioned in previous sections, and generate group parameters using the system parameters. This process has to be done once for each group, and it is not dependent on how complex the credentials to be issued are. We also measured unit times for updating a credential, which is used in user revocation. The unit time for updating a credential is dependent on the number of attributes to be updated. In our case, since we use credential updates just for user revocation, updating a single value will be sufficient. Therefore, the number of attributes for different complexities do not affect the unit times as in issuance. Moreover, upon closer inspection one can see that the increase in unit times for validating the proofs as the number of attributes increase is slower compared to proving the same number of attributes. This has two reasons. Firstly, validating attributes requires a constant setup time, which is the reason behind the fact that validating a simple proof is harder than proving one. Secondly, after the setup, the impact of the number of attributes on the compututation in validation process is less than the impact of them on the proving process. Therefore, the unit times of validation process are increasing slowlier compared to the proving process.

Table 5.2: The execution times for unit operations

	Simple	Medium	Complex
Issue Credential	0.84 seconds	1.14 seconds	2.04 seconds
Prove Attributes	0.07 seconds	0.18 seconds	0.56 seconds
Validate Attributes	0.13 seconds	0.18 seconds	0.52 seconds
Group Registration	16.4 seconds	16.4 seconds	16.4 seconds
Update Credential	0.65 seconds	0.65 seconds	0.65 seconds

5.2 Stress Tests

We applied simulations in order to test the efficiency of the entire system given the times of the unit operations. Since user registration takes place between the user and the group manager (issuer), and we aim to measure the performance of the verifier, we considered credential issuance (user registration) and group registration outside of the stress tests. As can be seen in Figure 3.1 the verifier and the issuer are different entities in the system and hence their load is not necessary dependent on each other.

As mentioned in previous chapters, we want to develop an environment that can run on mobile devices with low processing power. In the developed system, the Cloud will perform the computationally expensive tasks and leave as little work as possible to the client side. Client has to perform computations in two cases only: During the interaction with the issuer when creating credentials, and while creating proofs using the issued credentials. The issuance takes place only once and the client side does not perform less expensive computations compared to the issuer. The unit times for creating basic and medium proofs are also promising in the sense that they can be used in mobile devices with low-processing power. Moreover, considering the fact that client side cannot be used by more than one user at a time, performing a stress test to client side is unnecessary.

5.2.1 Simulation Model

We use M/D/m/m queueing system for the performance evaluation of the system. The first letter indicates that the nature of the arrival process is memoryless, which is a Poisson process. The second letter indicates the nature of the service times, which is deterministic in our case and can be found in the Table 5.2. The third letter indicates the number of servers, which differs from scenario to scenario. The last letter indicates that we use a finite queue. We consider a single queue with a new incoming request added at the end. Whenever a server becomes idle, it takes the the request at the top of the queue. The Figure 5.1 depicts our simulation model.

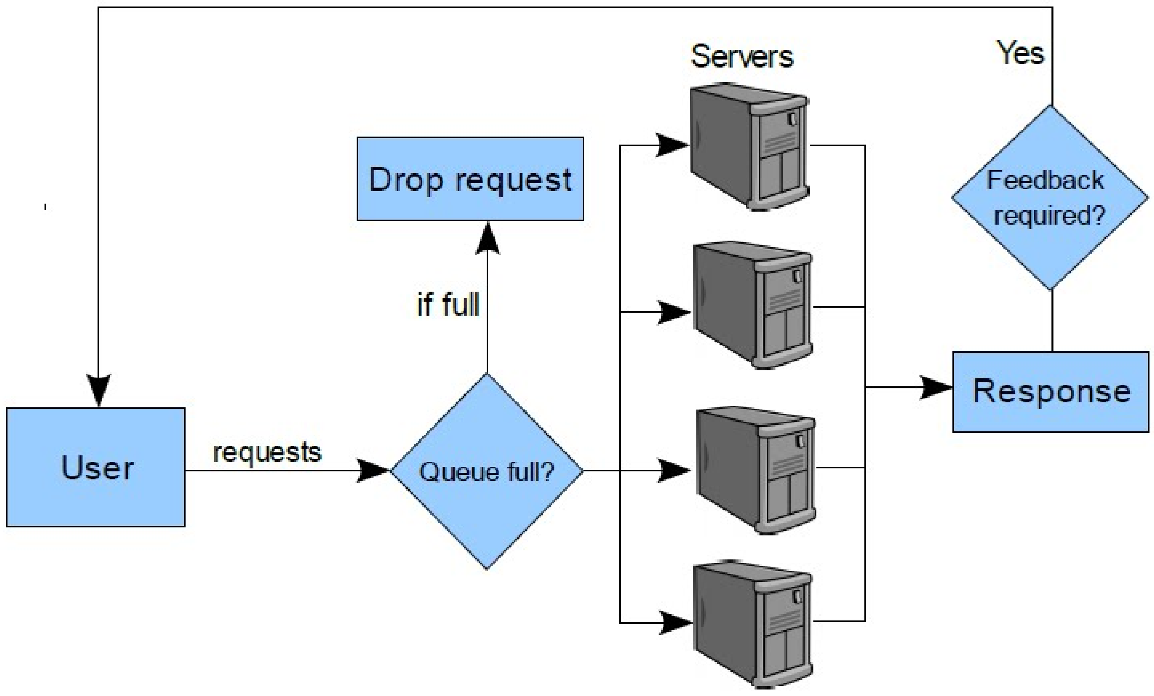


Figure 5.1: The simulation model

5.2.2 Simulation Parameters and Performance Metric

In this section, we present the variables in the simulation model and introduce the performance metric. The parameters considered are as follows.

- **Number of clients** : Number of clients that are connected to the server, denoted by N_c.
- **Number of servers** : Number of servers that are online, denoted by m. Since we perform a stress test on the verifier, throughout the section we will refer to server as a computer responsible for performing the duties of a verifier, unless specified otherwise.
- **Interarrival time of requests** : Interarrival time of requests is a exponentially distributed random variable. We denote the mean interarrival time of requests for each user by t, which is expressed in minutes. For example, if $t = 3$, each client makes one request in three minutes on average.
- **Complexity of proofs** : As mentioned in the Section 5.1, we consider three types of credentials which are 'simple', 'medium', and 'complex'. We denote them by c_1, c_2, and c_3 respectively. The type of a proof depends on how many attributes it contains and how many it reveals. The details can be seen in Table 5.1.
- **Number of permissions required** : Number of permissions in order to access Cloud service, it can take values one, three and six.
- **Session Duration** : The parameter specifying the duration of sessions in terms of minutes. When the specified number of minutes pass after a successful authentication, the user becomes unauthenticated.

We consider the *response time* as the main performance metric. We define *response time* as the time passed between the arrival of a request and its successful execution by the server. We do not consider any network delays in the model. However, in case a request becomes pending, we take into account the time spent in queue and the execution time. If the action indirectly requires authentication of some other members, we take as response time the time passed until all these processes are completed and approved.

In a probable use case scenario, once the user is authenticated, she needs no more cryptographic operations unless her authentication expires for some reason. We analyze three different scenarios. Firstly, we consider a variation of a worst case scenario, where every file needs only one permission share to be accessed, in order to see how the verifier behaves under increasing number of validation requests. For that, we assume that every request requires users to authenticate themselves. We perform various comparisons to see the full effects of changes in the number of servers, the mean interrival time of requests and the complexity of proofs. In addition to these performance evaluations, we limit the size of the queue and examine the throughput of the proposed system. In the second experiment, we assume that even the users that are required to give the permissions are not authenticated. Therefore, every request requires as many authentications as the number of permissions. In the third scenario, we consider a more realistic case and assume that every authentication lasts for a period of time. Whenever a request is sent by a group member, members are randomly chosen from the group until the minimum required number of shares are gathered. During this process, if any chosen member is unauthenticated, she goes through the authentication process first. In addition to these scenarios, we also examine the effect of user revocation on the average response time.

5.3 Simulation Results

In this section we provide the results of the simulation and performance analysis via performance charts.

5.3.1 Worst Case Average Response Time Analysis

Firstly, we measure the difference between the complexities of the proofs. In Figure 5.2, one can see differences between the response times for simple, medium and complex proofs, using one server and fixing the interarrival time to one minute. The figure shows that a single server can serve up to approximately 500 concurrent users with simple or medium complexity proofs with a response time less than one second. However, with complex proofs, the response time grows much faster with respect to the number of concurrent clients compared to simple and medium complexity proofs. Another observation is that the response times with simple and medium proofs behave linearly with respect to the number of concurrent clients, whereas the complex proofs have exponential behavior. If a provider wants to serve more than 200 concurrent clients using complex proofs, it has to use more than one server to achieve a response time under one second as we shall see later.

In Figure 5.3, we compare the average response times using different interarrival times: 20 seconds, one minute and three minutes. We apply the simulation with the medium credential using four servers. As expected, as the interarrival time decreases, the number of requests in a given time period increases and hence the response time gets longer. This shows the importance of scaling the number of servers with respect to user

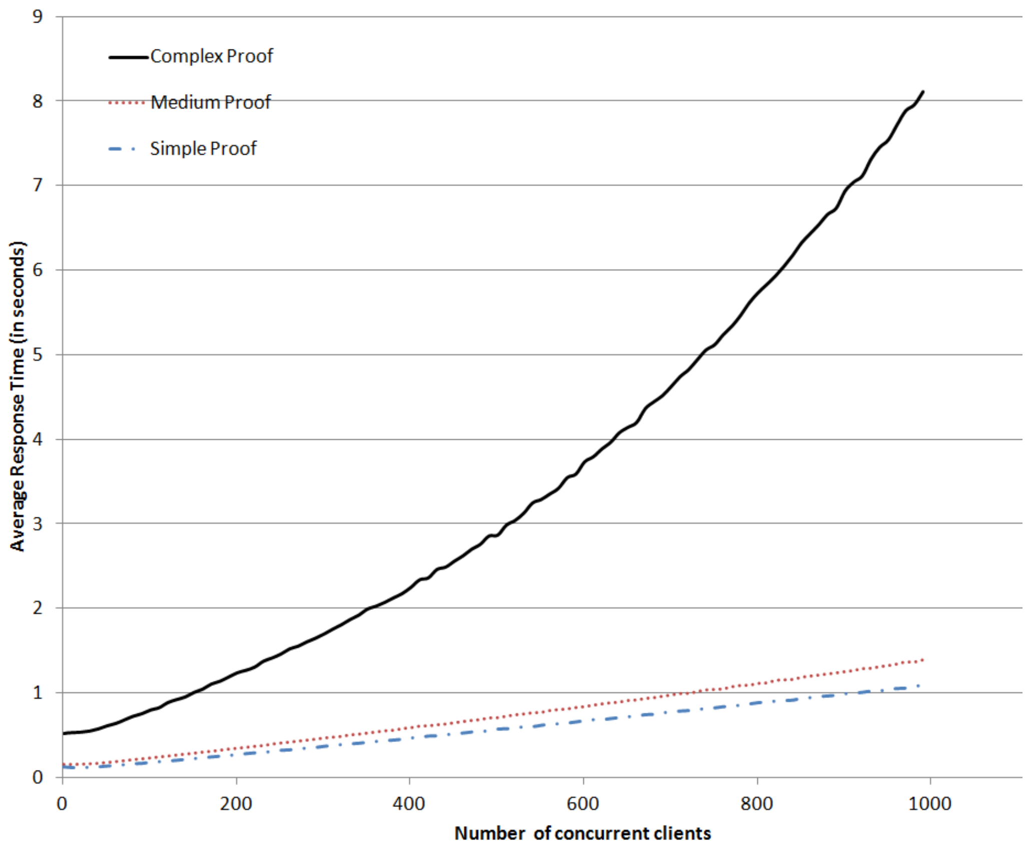


Figure 5.2: Response time analysis w.r.t. number of clients for different proof complexities, $m = 1, t = 1$

behaviour at the given time. Even with medium complexity proofs, if the users tend to send requests very frequently, the delay in response times increases up to more than one second after exceeding approximately 500 concurrent users.

We also comparatively analysed average response time when using different number of servers: 1, 2, 4, 8. Figure 5.4 shows the results of this analysis with the interarrival time of one minute using medium complexity proofs. The average response time decreases linearly with respect to the number of servers. As can be seen in Figure 5.4, using one server would delay the verification up to one second after 500 users. By using two servers, approximately 900 concurrent users can be handled with average response time under one second. The average response times when using four or eight servers are quite slowly increasing with respect to number of concurrent clients. Using four servers, a provider can serve up to 1000 concurrent users with quite acceptable response times, however the marginal gain is not so significant.

5.3.2 Ratio Comparison

As mentioned, the unit times are the measured times for indivisible operations. When we divide the actual time spent by the system by the unit time, we acquire the ratio between

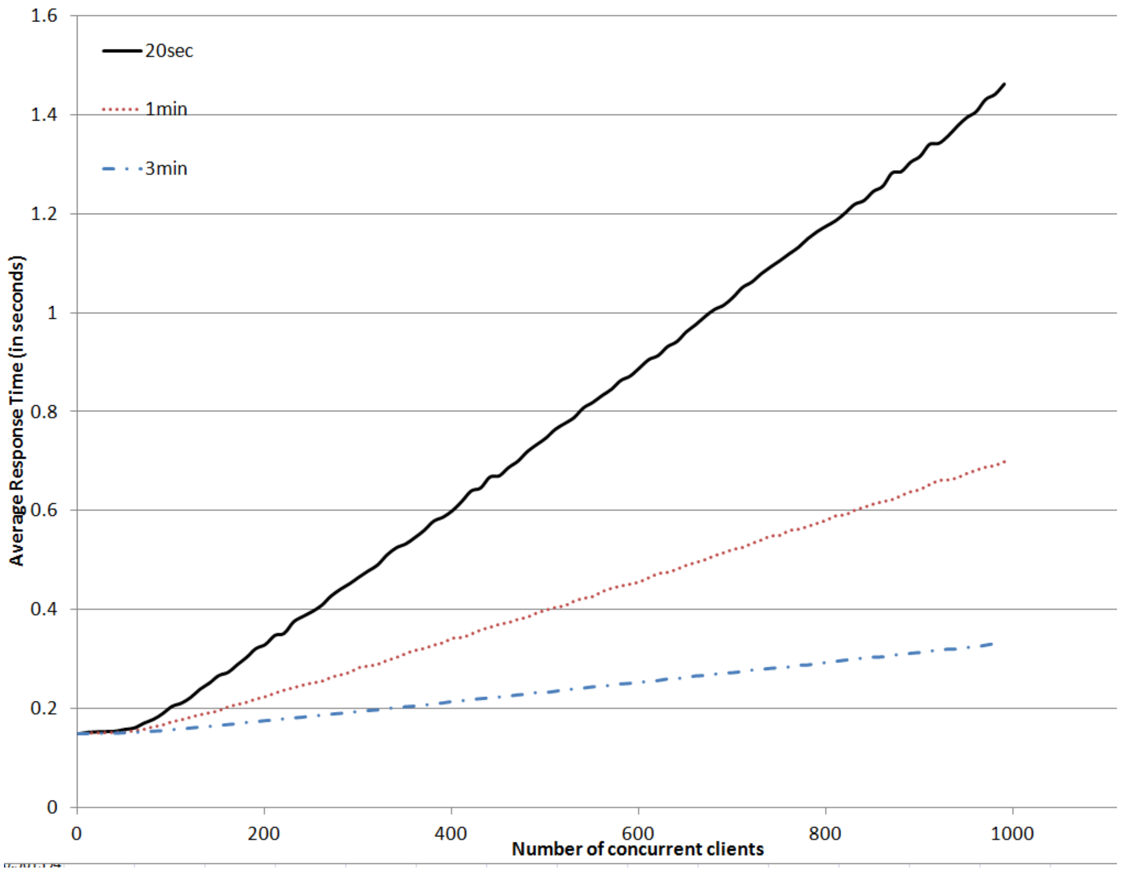


Figure 5.3: Response time analysis w.r.t. number of clients for different mean interarrival times using medium complexity proofs, $m = 2$

the actual times and the unit times. When there is only one user the actual times are equal to the unit times, so the ratio is equal to one. Hence, the ratio shows us the effect of serving concurrent users on the average response time. In Figure 5.5 one can see the difference between average ratios (response time / unit time) while using simple, medium and complex proofs. The average ratios tend to increase with a similar rate even though the number of attributes are different 5.1. Thus, we conclude that the unit times for validation of a proof are the primary factors in determining the average response time, since the average ratios under different complexities tend to behave similarly, while the average response times under different complexities increase at different rates, as can be seen in Figure 5.2.

5.3.2.1 Throughput Analysis

As mentioned before, we consider an infinite queue throughout the simulation. However, we also examine the effect the queue size would have in case it is finite. We define throughput as the proportion of successful jobs to total number of jobs. To examine the throughput of the system, we try different queue sizes and calculate the expected throughput of the system using different proof complexities when there are 1000 concurrent users

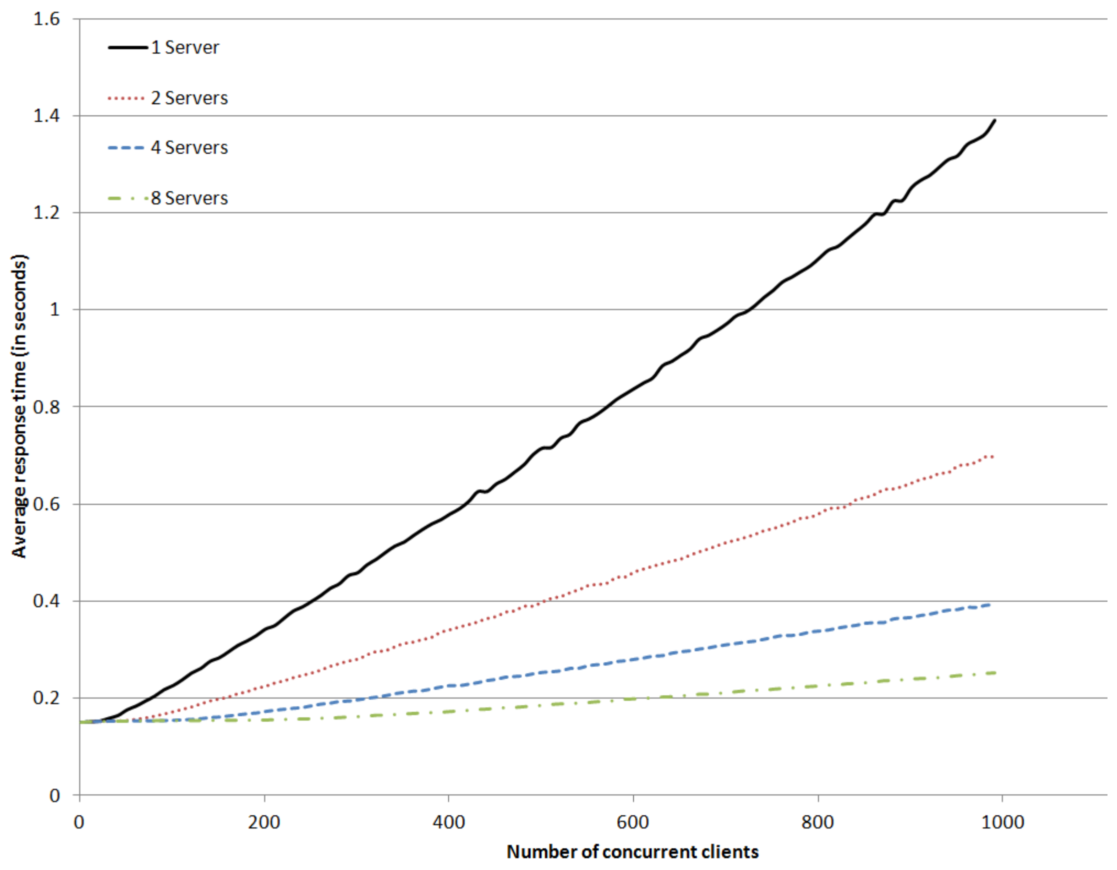


Figure 5.4: Comparative response time analysis w.r.t. number of clients for different servers using medium complexity proofs, $t = 1$

with the mean interarrival time of one minute . As expected, when using complex proofs, throughput is approaching one slowlier compared to the simpler proofs. The Figure 5.5 shows the throughput reaches one, even with the clients using complex proofs, when using a queue of size larger than 100.

5.3.3 Worst Case Average Response Time Analysis Using Different Number of Required Permissions

Once an action request arrives, the verifier waits until the required number of permissons are gathered in order to take the requested action. In this case, we assume the worst, meaning that every action requires an authentication every time. Doing so, we examine the system performance in the worst case. Figure 5.7 shows the comparison between different number of required permissions, when it takes values one, three and six respectively. When the number of required permissions is six, the increase in the average response time with respect to the number of concurrent clients is exponential, and the average response time becomes larger than one second with 800 concurrent clients. Figure 5.7 also shows that the verifier can serve up to 1000 clients with four servers with an average response

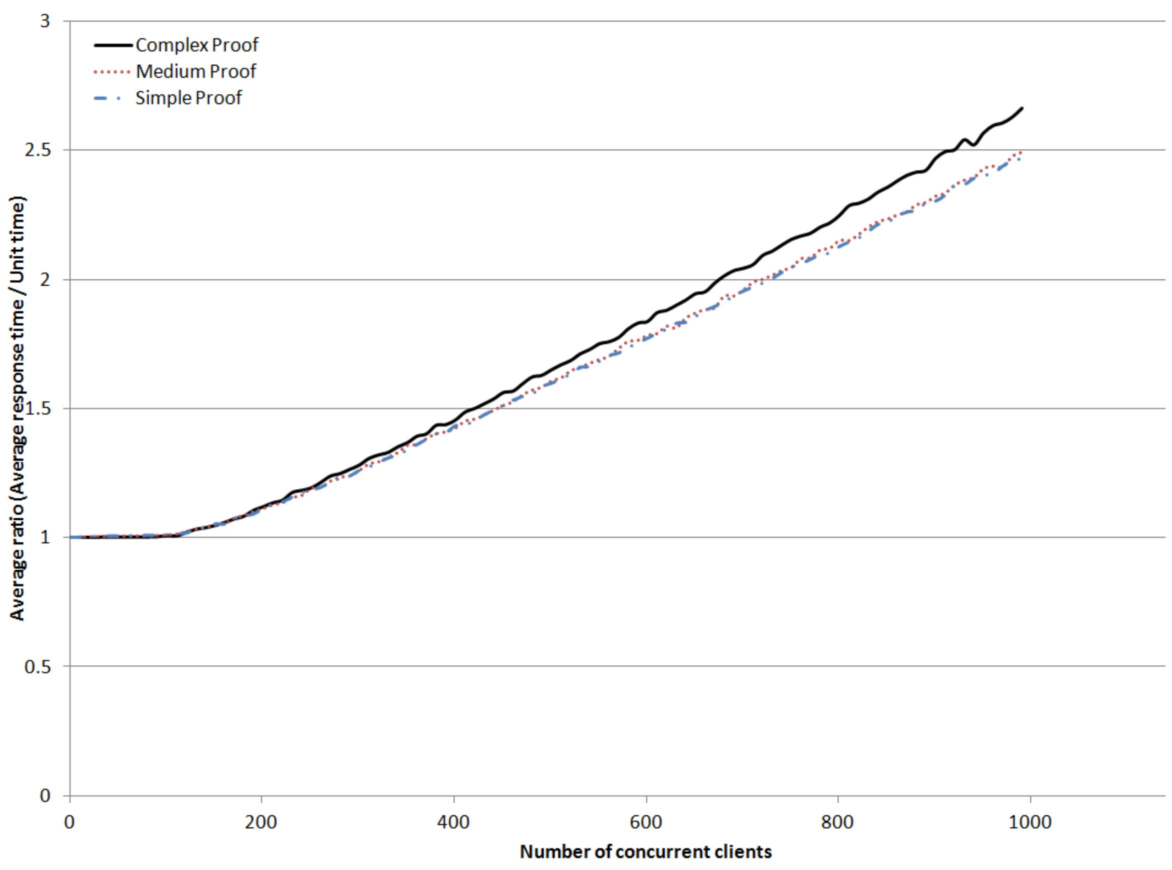


Figure 5.5: [Response time / Unit time] ratio analysis w.r.t. number of clients for different proof complexities, c_m, $m = 4$, $t = 1$

time under one second, when required number of permissions is equal to three.

5.3.4 Average Response Time Analysis Using Authenticated Sessions

Table 5.3: Groups and Roles

	Small	Medium	Large
Junior Programmers	0	0	7
Senior Programmers	5	8	5
Testers	3	5	7
Supervisors	1	1	5
Managers	1	1	1
Total	10	15	25

We also consider a realistic scenario, where users need to authenticate themselves once and during a pre-determined session they do not need to authenticate themselves again. After the authentication expires, they have to renew the authentication. During authentication, the user directly communicates with the Cloud service provider, and hence

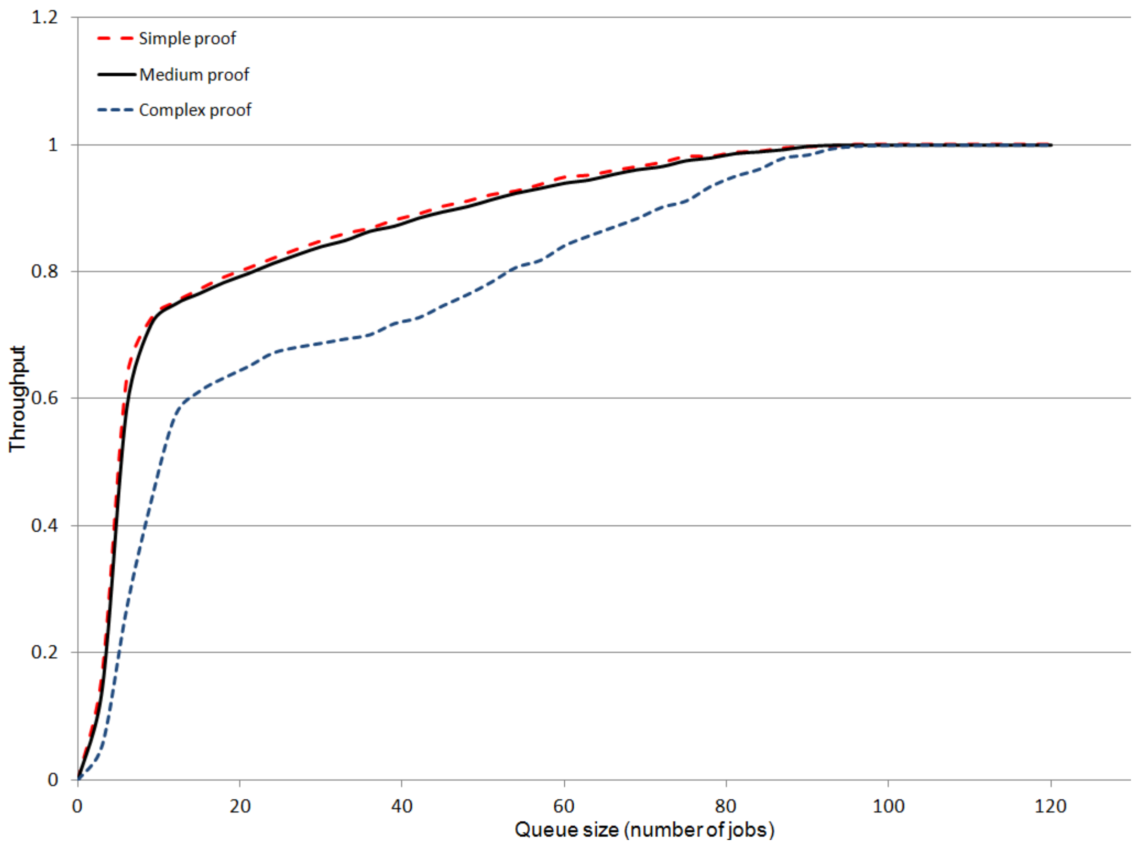


Figure 5.6: Throughput analysis w.r.t. number of clients for different proof complexities, $m = 4$, $t = 1$

has no cost to the verifier other than a simple authentication check (whether she has an active session or not). In this scenario, we also consider different groups with varying number of members and roles. The distribution of the roles and the number of users can be found in Table 5.3. Moreover, we take into account that the actions may require different number of permissions in order to be executed. The shares held by the members according to their roles can be found in Table 5.4.

As expected, the average response times are very low compared to the worst case analyses. This is very promising and shows that the system is very efficient even with a session duration of 10 minutes. The Figure 5.9 shows the comparison between the different proof complexities, and the Figure 5.8 shows the comparison between sessions with different durations. The required number of shares are fixed to six in both cases. However, the results also show that the effect of this constraint is very small. Since the users tend to request actions at least once in a minute, the authentication required in File Access Protocol lasts also for at least ten minutes and hence, does not have an extra cost.

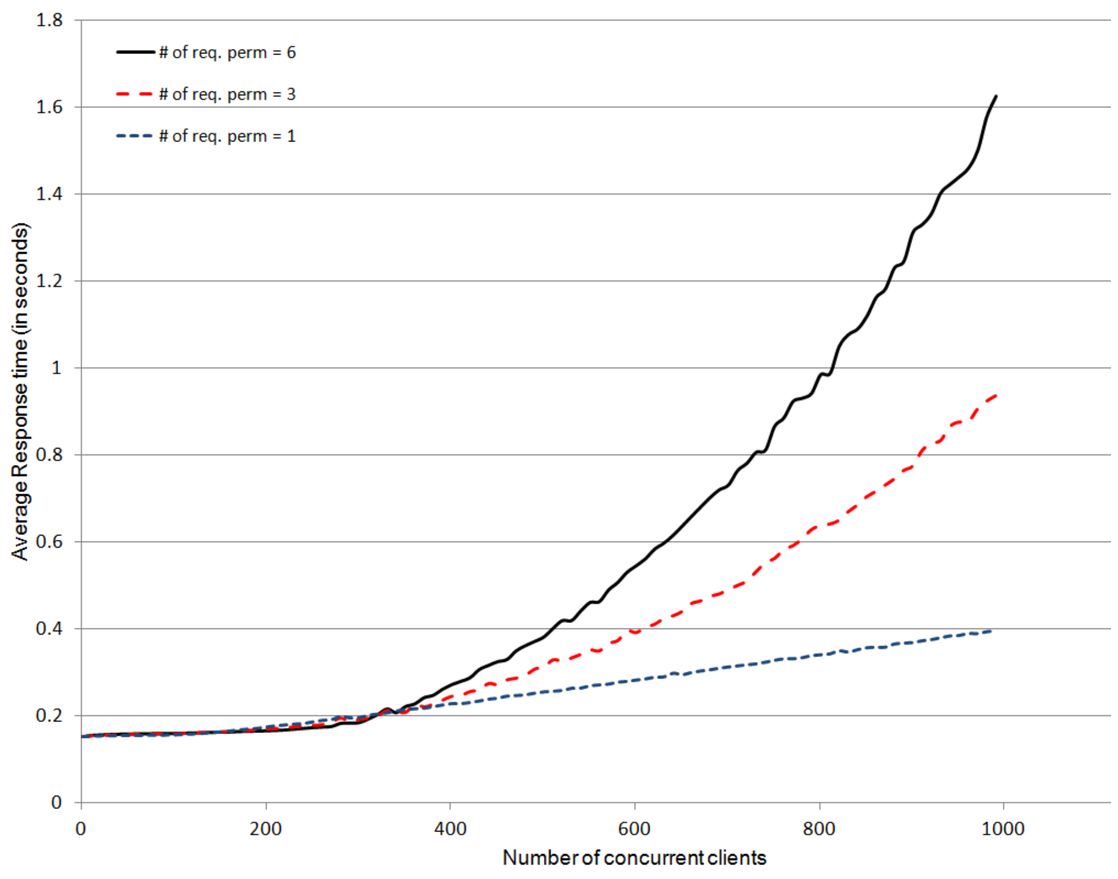


Figure 5.7: Worst Case Average response time analysis w.r.t. number of clients for different number of required permissions using medium complexity proofs, $m = 4$, $t = 1$

Table 5.4: The distribution of permission shares

	Number of Shares
Junior Programmers	1
Senior Programmers	2
Testers	1
Supervisors	4
Managers	6

5.3.5 Average Response Time Analysis In Case of Revocation

Since the revocation process concerns groups, we analyze the change in response times group by group instead of considering all the clients together. The change in average response time for a single group is marginal compared to the overall average. Hence, considering all the clients hide the actual increase in the average response time for single groups. We evaluate the change in average response time with respect to the system time and analyse how the average response times behave in case of revocation. As expected, the average response times increase as the first request arrives. In Figure 5.10 the first

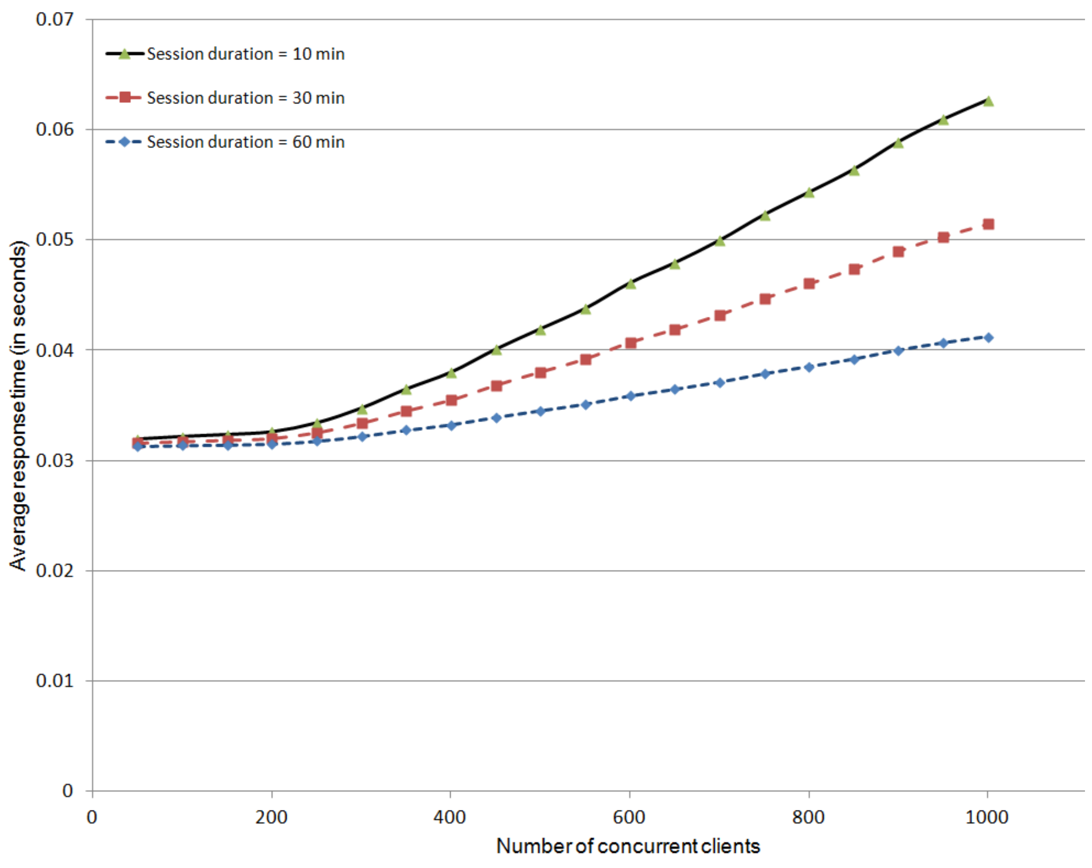


Figure 5.8: Average response time analysis w.r.t. number of clients for different session durations, c_m, $m = 4$, $t = 1$

request after a revocation arrives at 90^{th} second, and in Figure 5.11 at 150^{th} second. From that point on, it starts to decrease, since the group members, who already updated their credentials, have no more expensive operations. In Figure 5.10, the mean interarrival time is one minute, therefore, after one and a half minutes, the group members will have finished their update processes and the system reaches steady state. Figure 5.11 shows the case when the mean interarrival time is three minutes. When we compare two figures, we can observe that the time spent to reach steady state behaves linearly with respect to mean interarrival time, which is an expected result since only the first operation after revocation is expensive and the mean interarrival time dictates the arrival of the first operation. Another observation is that the average response time decreases quicklier in small groups compared to the large groups. This is the result of the fact that in large groups, it is more probable that there is at least one user that has not yet issued any requests up until a certain point in time.

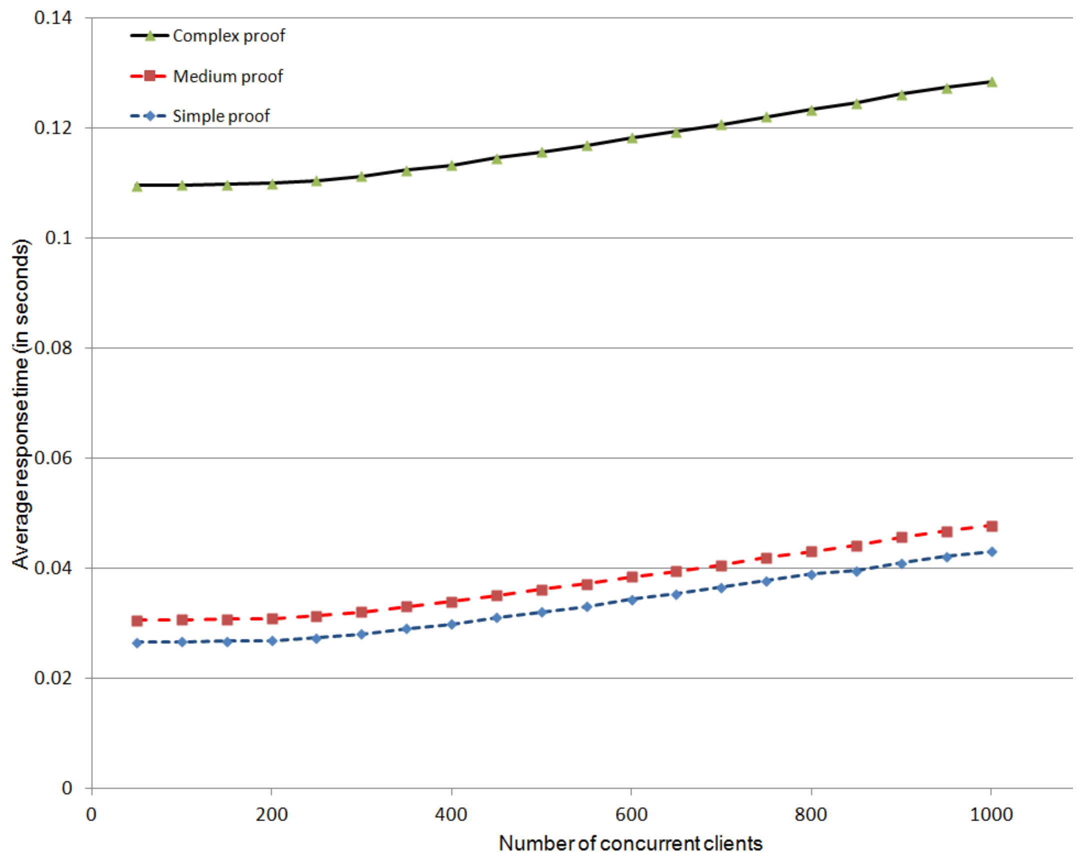


Figure 5.9: Average response time analysis w.r.t. number of clients for different proof complexities, $session = 10min$, $m = 4$, $t = 1$

5.4 Discussion

After running simulations with different parameters, we conclude that using complex credentials is an extra burden for the provider. At least two attributes (group identification number and role within the group) are necessary to define a proper access right scheme. However, some groups may want to form the access right scheme in a more detailed and enhanced way. To achieve this, they need extra attributes, and hence more complex proofs. Moreover, keeping in mind that the interarrival time can change from time to time regardless of the group, the server needs to observe the active behaviour in order to adjust the number of servers. The results are promising and show us that the proposed system can easily maintain up to 1000 concurrent users with interarrival time of one minute by using four servers. Moreover, the averager response time analysis using sessions show us the efficiency of the proposed system is very promising, and shows that with season duration of 30 minutes can handle 1000 concurrent users with a very quick response time under tenth of a second. Finally, the average response time analysis in case of revocation shows the system can maintain revocation efficiently, too. The response times may increase during credential updates, however after updates are finished, the system reaches steady

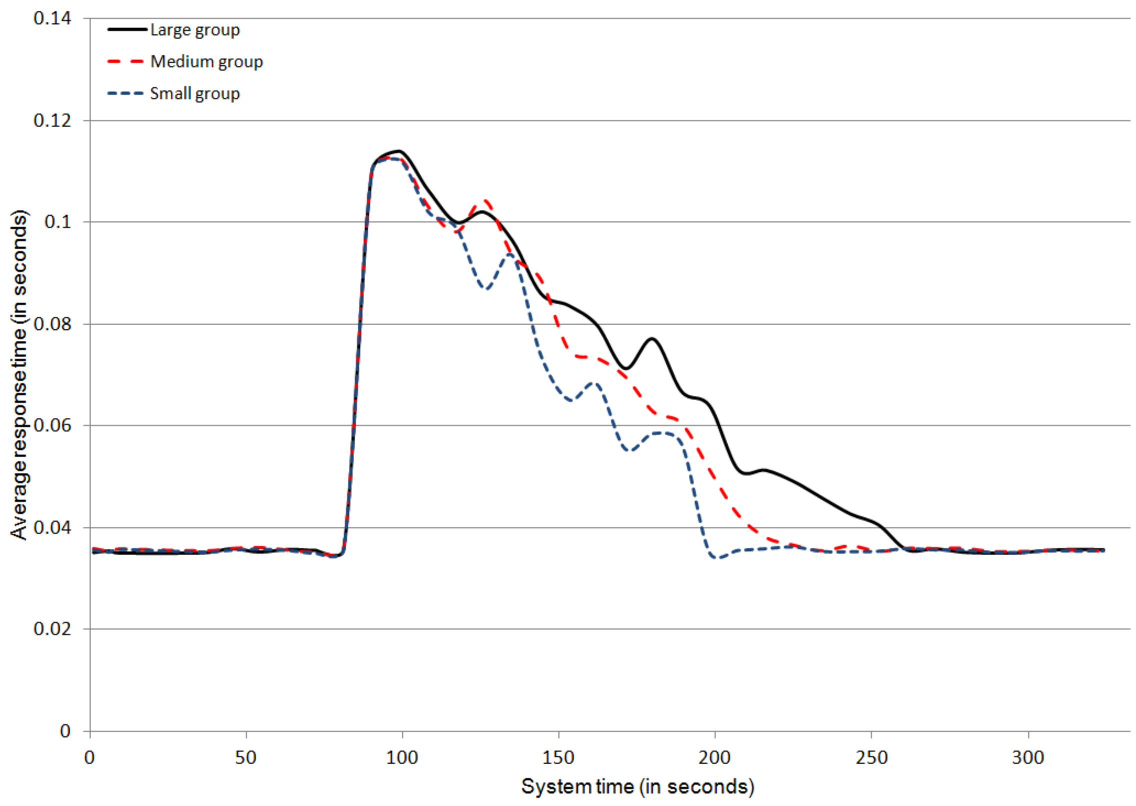


Figure 5.10: Average response time analysis w.r.t. system time for different sizes of groups using medium complexity proofs, $m = 1$, $t = 1$

state again.

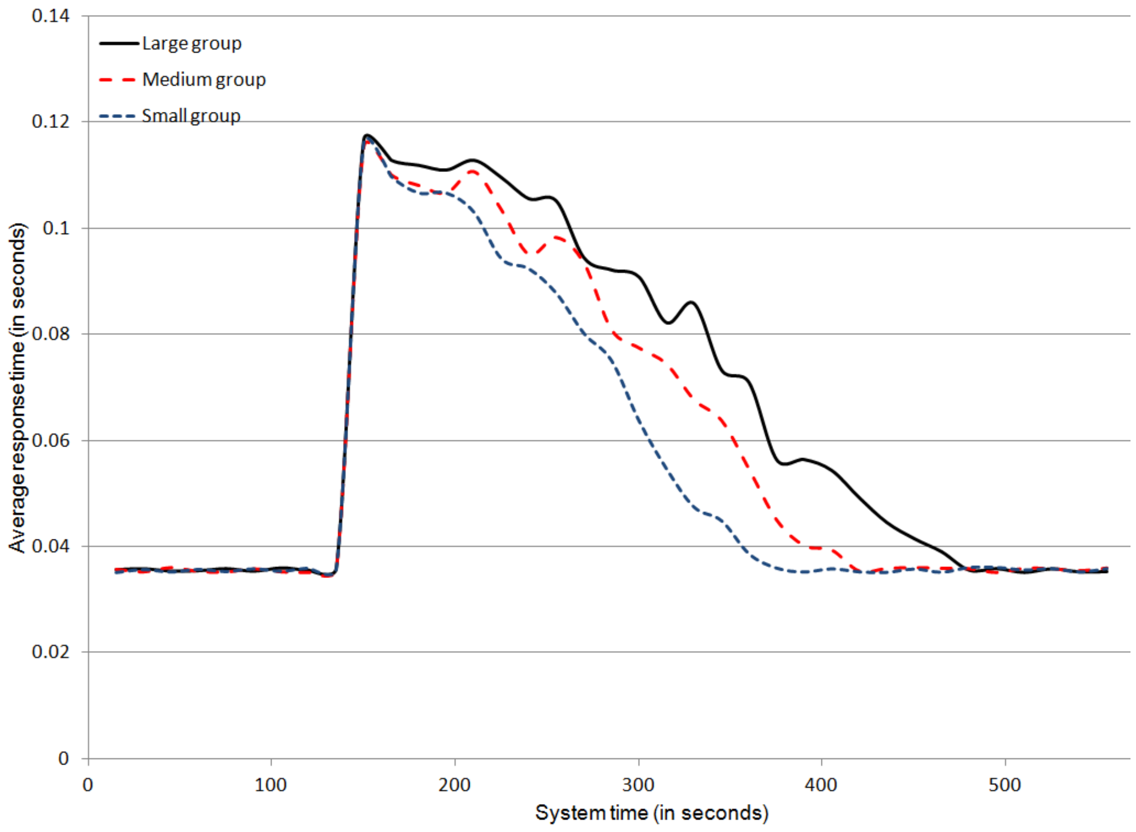


Figure 5.11: Average response time analysis w.r.t. system time for different sizes of groups, using medium complexity proofs, $m = 1, t = 3$

Chapter 6

Conclusion and Future Work

The main motivation of this thesis was to use the advantages of Cloud computing paradigm in a software development scenario while providing security and a novel approach to access right scheme using group signatures. While we achieved to implement a proof-of-concept application, a more sophisticated version of the same idea may be implemented for an industrial use. Considering the increasing popularity of Cloud computing, such a system would be beneficial to software development groups. Using an anonymous credential system like *idemix*, and hence using the group signature schemes, seems to be an efficient way to implement access rights in a secure way. By using the CL signature scheme, which relies on zero-knowledge proofs, we also provided anonymity for group members against Cloud. From the Cloud point of view, users have roles that are issued by their manager and Cloud cannot possess any more knowledge than the roles dealt by the group manager.

In this thesis, we proposed a secure software development environment in Cloud which supports software development groups that have different kinds of hierarchies. We have done simulation-based performance evaluation of the system to measure the response times under different scenarios. In order to do that, we applied a simulation using M/D/m/m queues. The main features of the system can be summarized as follows:

- By taking advantage of Cloud computing, we only placed the light cryptographic operations on the client side. That way, we encouraged the use of the mobile devices with low processing power, enhancing the mobility of the system.
- We used the *idemix* library to ensure secure connection between the system users and the system. By defining roles and adopting an access right scheme approach during the group registration, we freed the server from dealing with the inner workings of groups. Furthermore, we also preserved the anonymity of the users.

- We did a performance evaluation using an M/D/m/m simulation model to figure out how efficient the system is. The results are promising and shows that the client side does not have to do expensive computations, and that the server is capable of maintaining response time under 1 second while serving 1000 clients with 4 servers. In the scope of the thesis, we have developed a prototype level implementation.

As a future work our proposed system can be implmented as a real SaaS to operated on Cloud. By utilizing the advantages of Cloud computing, the proposed system also offers easy maintainability and reduces the risk of data loss by centralizing the data and processing power. Moreover, it enables easy use of versioning systems. The system can be easily run in mobile devices, once the dependencies to third parties are removed.

Bibliography

[1] Amazon Elastic Compute Cloud. Retreived January 26, 2012, from `http://aws.amazon.com/ec2/`.

[2] Amazon Web Services. Retreived January 26, 2012, from `aws.amazon.com`.

[3] Dropbox. Retreived January 26, 2012, from `http://www.dropbox.com`.

[4] Gmail. Retreived January 26, 2012, from `http://mail.google.com`.

[5] GoogleAppEngine. Retreived January 26, 2012, from `code.google.com/appengine`.

[6] GoogleCode. Retreived January 26, 2012, from `code.google.com`.

[7] GoogleDocs. Retreived January 26, 2012, from `docs.google.com`.

[8] Microsoft Azure. Retreived January 26, 2012, from `http://www.windowsazure.com`.

[9] *Secure hash standard.* National Institute of Standards and Technology, Washington, 1995. Note: Federal Information Processing Standard 180-1.

[10] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[11] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880, pages 255–270. Springer, 2000.

[12] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In *Proceedings on Advances in Cryptology*, CRYPTO '88, pages 37–56, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[13] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: how to remove intractability assumptions. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, STOC '88, pages 113–131, New York, NY, USA, 1988. ACM.

[14] P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi. Get shorty via group signatures without encryption. In *Proceedings of the 7th International Conference on Security and Cryptography for Networks*, SCN'10, pages 381–398, Berlin, Heidelberg, 2010. Springer-Verlag.

[15] E. Bresson and J. Stern. Efficient revocation in group signatures. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, PKC '01, pages 190–206, London, UK, 2001. Springer-Verlag.

[16] J. Camenisch and A. Lysyanskaya. A signature scheme with efficient protocols. In *Proceedings of the 3rd International Conference on Security in Communication Networks*, SCN'02, pages 268–289, Berlin, Heidelberg, 2003. Springer-Verlag.

[17] J. Camenisch and M. Michels. A group signature scheme based on an RSA-variant. Technical Report RS-98-27, BRICS, Dept. of Computer Science, University of Aarhus, Nov 1998.

[18] J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 410–424, London, UK, 1997. Springer-Verlag.

[19] J. Camenisch and E. Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, pages 21–30, New York, NY, USA, 2002. ACM.

[20] D. Chaum and H. van Antwerpen. Undeniable signatures. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 212–216, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[21] D. Chaum and E. van Heyst. Group signatures. In *Advances in Cryptology - EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 257–265. Springer-Verlag, 1991.

[22] L. Chen and T. P. Pedersen. New group signature schemes (extended abstract). In *Advances in Cryptology - EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 171–181. Springer-Verlag, 1995.

[23] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 85–90, New York, NY, USA, 2009. ACM.

[24] M. Christodorescu, R. Sailer, D. L. Schales, D. Sgandurra, and D. Zamboni. Cloud security is not (just) virtualization security: a short paper. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 97–102, New York, NY, USA, 2009. ACM.

[25] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[26] U. Feige and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, STOC '90, pages 416–426, New York, NY, USA, 1990. ACM.

[27] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in Cryptology—CRYPTO '86*, pages 186–194, London, UK, 1987. Springer-Verlag.

[28] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38:690–728, July 1991.

[29] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18:186–208, February 1989.

[30] T. C. Lethbridge and R. Laganiere. Basing software development on reusable technology. In K. Mosman, editor, *Object-Oriented Software Engineering : Practical Software Development using UML and Java*, pages 91–102. McGraw-Hill Education, 2005.

[31] P. Mell and T. Grance. The NIST definition of cloud computing. Technical Report SP - 800 - 145, National Institute of Standards and Technology, 2009. [Stand: 16.03.2011].

[32] J. J. Quisquater, L. Guillou, M. Annick, and T. Berson. How to explain zero-knowledge protocols to your children. In *Proceedings on Advances in Cryptology*, CRYPTO '89, pages 628–631, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[33] IBM Research Zurich Security Team. Specification of the identity mixer cryptographic library. IBM Research Report 3730, IBM Research, Apr 2010.

[34] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning. Managing security of virtual machine images in a cloud environment. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 91–96, New York, NY, USA, 2009. ACM.

[35] X. Zhang, J. Schiffman, S. Gibbs, A. Kunjithapatham, and S. Jeong. Securing elastic applications on mobile devices for cloud computing. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 127–134, New York, NY, USA, 2009. ACM.