



UNIVERSITY OF BEIRA INTERIOR  
Department of Computer Science

# Digital Certificates and Threshold Cryptography

Adolfo Barbosa do Amaral Peixinho

Thesis submitted to the Department of Computer Science for the fulfillment of the requirements  
for the degree of Master in Science made under the supervision of **Doctor Professor Paul  
Andrew Crocker**, from the Department of Computer Science of University of Beira Interior,  
Covilhã - Portugal.

Covilhã, October 19, 2013



*“Beware the Four Horsemen of the Information Apocalypse: terrorists, drug dealers, kidnappers, and child pornographers. Seems like you can scare any public into allowing the government to do anything with those four”*

---

Bruce Schneier

*“I think that I am among the few lucky ones who are exploiting complexity. Most people are unhappy with the emergence of complexity, they would prefer it if the world were very simple, but then it would be a doom for a cryptographer like myself”*

---

Adi Shamir

*“If privacy is outlawed, only outlaws will have privacy.”*

---

Phill Zimmerman





# Acknowledgements

I dedicate a sincere thanks to all who made possible the conclusion of this project.

I would also like to thank Professor Paul Crocker and Professor Simão Sousa for their guidance and valuable information conveyed, for the precise and clear way of transmitting wisdom and scientific knowledge, for their availability and above all for their teaching and advice.

I thank my family, in especially my mother, for she always backed me up throughout my life.

Last but not least, I would like to thank Francisca Joana Neves, for the patience and support.

Thank you!



# Abstract

This dissertation discusses the use of *secret sharing* cryptographic protocols for distributing and sharing of secret documents, in our case PDF documents.

We discuss the advantages and uses of such a system in the context of collaborative environments.

Description of the cryptographic protocol involved and the necessary Public Key Infrastructure (PKI) shall be presented. We also provide an implementation of this framework as a “proof of concept” and fundament the use of a *certificate extension* as the basis for *threshold cryptography*.

Details of the *shared secret* distribution protocol and *shared secret* recovery protocol shall be given as well as the associated technical implementation details.

The actual *secret sharing* algorithm implemented at this stage is based on an existing well known *secret sharing scheme* that uses polynomial interpolation over a finite field.

Finally we conclude with a practical assessment of our prototype.

**Keywords:** Secret Sharing, Threshold Cryptography, Public Key Infrastructure, certificate extension.



# Contents

<b>Acknowledgements</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xi</b>
<b>List of Algorithms</b>	<b>xiii</b>
<b>Acronyms</b>	<b>xv</b>
<b>Glossary</b>	<b>xvii</b>
<b>Notation</b>	<b>xix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation and Objectives . . . . .	2
1.2 Contributions . . . . .	2
1.3 Thesis Outline . . . . .	3
<b>2 State of the Art</b>	<b>5</b>
2.1 Secret Sharing Schemes . . . . .	5
2.2 Threshold Secret Sharing Scheme . . . . .	5
2.2.1 Shamir's Secret Sharing Scheme . . . . .	5
2.2.2 Blakley Secret Sharing Scheme . . . . .	7
2.3 Access Structures . . . . .	8
2.4 Overview of Secret Sharing Schemes . . . . .	9
2.5 Evolution of Public-Key Infrastructure . . . . .	12
2.5.1 Digital Certificate Extensions . . . . .	14
2.6 Related Work . . . . .	14
<b>3 Requirements and Features</b>	<b>15</b>
3.1 Preliminaries . . . . .	15
3.1.1 Object Identifier (OID) . . . . .	15
3.1.2 Abstract Syntax Notation One (ASN.1) . . . . .	15
3.1.3 One-Way Functions . . . . .	17
3.1.4 Pseudo Random Number Generators . . . . .	18
3.1.5 Mutual Authentication . . . . .	20
3.1.6 SCrypt Password Encryption . . . . .	21
3.2 Framework Requirements . . . . .	22
3.3 PKI CA Software . . . . .	23
3.3.1 EJBCA Features . . . . .	23
3.3.2 EJBCA Requirements . . . . .	25
3.4 Conclusions . . . . .	25

<b>4</b>	<b>Architecture</b>	<b>27</b>
4.1	Communication Model between CA and Web Application . . . . .	27
4.2	Secret Distribution . . . . .	30
4.3	Secret Recovery . . . . .	32
<b>5</b>	<b>Implementation</b>	<b>35</b>
5.1	EJBCA Deployment Process . . . . .	35
5.1.1	Securing MySQL <i>Relational Database Management System</i> (RDBMS) . . .	35
5.1.2	Securing JBoss Server . . . . .	36
5.1.3	EJBCA Installation . . . . .	36
5.1.4	CA Hierarchy Implementation . . . . .	38
5.2	Web Application Technologies . . . . .	41
5.2.1	Securing GlassFish Server . . . . .	41
5.2.2	Framework on the Web . . . . .	42
5.2.3	Web Application Workflow . . . . .	43
<b>6</b>	<b>Security Analysis</b>	<b>53</b>
6.1	Optimization and Obsfuscation . . . . .	53
6.1.1	ProGuard . . . . .	53
6.2	SSL-TLS Audit . . . . .	54
6.2.1	Qualys SSL-TLS Server Rating Guide . . . . .	54
6.2.2	JBoss 5.1 SSL-TLS Audit . . . . .	56
6.2.3	Glassfish 3.1.2 SSL-TLS Audit . . . . .	58
6.2.4	SSL-TLS Audits on other Organizations . . . . .	60
6.2.5	Conclusions . . . . .	61
6.3	Attack Trees . . . . .	61
6.3.1	Models of Attacks on the Framework . . . . .	62
6.4	Conclusions . . . . .	65
<b>7</b>	<b>Conclusions and Further Work</b>	<b>67</b>
7.1	Conclusions . . . . .	67
7.2	Future Work . . . . .	68
<b>A</b>	<b>Annexes</b>	<b>69</b>
A.1	EJBCA Configuration Property Files . . . . .	69
A.1.1	Certificate Store Configuration . . . . .	69
A.1.2	CRL Store Configuration . . . . .	69
A.1.3	Database Configuration . . . . .	69
A.1.4	EJBCA Configuration . . . . .	69
A.1.5	OCSP Configuration . . . . .	69
A.1.6	EJBCA Install Configuration . . . . .	70
A.1.7	Web Configuration . . . . .	70
A.1.8	Certificate Extensions Configuration . . . . .	71
A.2	Diagrams . . . . .	71
A.2.1	Use Case Diagrams . . . . .	72
A.2.2	Sequence Diagrams . . . . .	75
A.2.3	Activity Diagrams . . . . .	76
A.2.4	Deployment Diagrams . . . . .	77

A.2.5 DataBase Entity Relationship Diagram . . . . .	78
<b>Bibliography</b>	<b>79</b>





# List of Figures

2.1	Blakley Secret Sharing Scheme for threshold $k = 2$ .	7
3.1	OID tree	16
3.2	Tag, Length, Value triplet element.	16
3.3	Tag, Length, Value triplet of a triplet.	17
3.4	Examples of encoded length and value bytes	17
3.5	Random and Pseudorandom Number Generators (Stall)	19
3.6	Client certificate-based mutual authentication sequence.	21
4.1	Communication model between EJBCA-CA and the Web application using the Web service.	29
4.2	Process that leads to the creation of a secret distribution job.	30
4.3	Process that leads to the creation of a secret recovery job.	32
5.1	Certificate extension illustration.	37
5.2	EJBCA hierarchy diagram	39
5.3	Web application main window	43
5.4	Firefox browser warning	43
5.5	EJBCA public site	44
5.6	EJBCA certificate enrollment/request form	44
5.7	Web application login request	45
5.8	Web application login form	45
5.9	Web application login success	46
5.10	Web application group create	46
5.11	Web application group join	47
5.12	Web application group leave	47
5.13	Web application propose PDF	48
5.14	Web application get PDF file dialog	48
5.15	Web application secret distribute intent	49
5.16	Web application secret distribution initiated	49
5.17	Web application secret recover intent	50
5.18	Web application secret recover initiated	50
5.19	Web application show PDF document	51
5.20	Web application recovered PDF document	51
6.1	Proguard optimization-obfuscation process	53
6.2	JBoss 5.1 SSL-TLS score	56
6.3	JBoss 5.1 SSL-TLS supported protocols	56
6.4	JBoss 5.1 SSL-TLS supported protocol details	57
6.5	JBoss 5.1 SSL-TLS certification path	57
6.6	Glassfish 3.1.2 SSL-TLS score	58
6.7	Glassfish 3.1.2 SSL-TLS supported protocols	58
6.8	Glassfish 3.1.2 SSL-TLS supported protocol details	59
6.9	Glassfish 3.1.2 SSL-TLS certification path	59
6.10	Caixa Geral de Depósitos SSL-TLS score	60

6.11 University of Beira Interior SSL-TLS score . . . . .	60
6.12 Qualys-SSL Labs SSL-TLS score . . . . .	60
6.13 Attack Tree - Open Safe. Adapted from (Sch). . . . .	61
6.14 Attack Tree to forge a certificate request . . . . .	63
6.15 Attack Tree to steal a certificate . . . . .	64
A.1 UML diagrams overview. . . . .	71
A.2 Use case diagram of group management system . . . . .	72
A.3 Use case diagram of group secrets management system . . . . .	72
A.4 Use case diagram of the jobs/tasks management system . . . . .	73
A.5 Use case diagram for access to the private area of the Web application . . . . .	73
A.6 Use case diagram of the user edit system . . . . .	74
A.7 Use case diagram of the certificate request system . . . . .	74
A.8 Sequence diagram for user login . . . . .	75
A.9 Sequence diagram for user register . . . . .	75
A.10 Activity diagram of the distribute process . . . . .	76
A.11 Activity diagram of the recover process . . . . .	76
A.12 Deployment diagram of the framework . . . . .	77
A.13 Entity Relationship Diagram of the Web application database . . . . .	78

# List of Tables

2.1	Extensions of $(k, n)$ - Threshold Secret Sharing Schemes . . . . .	9
2.2	Various implementations of Secret Sharing Schemes . . . . .	11
3.1	Comparison between <i>Certification Authority (CA)-Public Key Infrastructure (PKI)</i> software . . . . .	23
6.1	Letter grade translation from numerical score . . . . .	55
6.2	Score percentage of each category . . . . .	55



# List of Algorithms

1	Secret sharing algorithm . . . . .	31
2	Secret recovery algorithm . . . . .	33



# Acronyms

<b>AIA</b>	<i>Authority Information Access</i>
<b>API</b>	<i>Application Programming Interface</i>
<b>ASN.1</b>	<i>Abstract Syntax Notation One</i>
<b>BER</b>	<i>Basic Encoding Rules</i>
<b>CA</b>	<i>Certification Authority</i>
<b>CER</b>	<i>Canonical Encoding Rules</i>
<b>CMP</b>	<i>Certificate Management Protocol</i>
<b>CRL</b>	<i>Certificate Revocation List</i>
<b>CSPRNG</b>	<i>Cryptographic Secure Pseudo Random Number Generator</i>
<b>CSR</b>	<i>Certificate Signing Request</i>
<b>CVE</b>	<i>Common Vulnerabilities and Exposures</i>
<b>DER</b>	<i>Distinguished Encoding Rules</i>
<b>EJB</b>	<i>Enterprise JavaBeans</i>
<b>EJBCA</b>	<i>Enterprise Java Beans Certificate Authority</i>
<b>ERD</b>	<i>Entity Relationship Diagram</i>
<b>GUI</b>	<i>Graphical User Interface</i>
<b>JCE</b>	<i>Java Cryptography Extension</i>
<b>JDK</b>	<i>Java Development Kit</i>
<b>JEE</b>	<i>Java Enterprise Edition</i>
<b>JKS</b>	<i>Java KeyStore</i>
<b>JSE</b>	<i>Java Standard Edition</i>
<b>JSF</b>	<i>Java Server Faces</i>
<b>LDAP</b>	<i>Lightweight Directory Access Protocol</i>
<b>OCSP</b>	<i>Online Certificate Status Protocol</i>
<b>OID</b>	<i>Object Identifier</i>
<b>PEM</b>	<i>Privacy-enhanced Electronic Mail</i>
<b>PER</b>	<i>Packed Encoding Rules</i>
<b>PKCS</b>	<i>Public Key Cryptographic Standard</i>
<b>PBKDF</b>	<i>Password-Based Key Derivation Function</i>
<b>PKI</b>	<i>Public Key Infrastructure</i>
<b>PRNG</b>	<i>Pseudo Random Number Generators</i>

<b>QSSS</b>	<i>Quantum Secret Sharing Schemes</i>
<b>RA</b>	<i>Registration Authority</i>
<b>RDBMS</b>	<i>Relational Database Management System</i>
<b>SSL</b>	<i>Secure Socket Layer</i>
<b>SSS</b>	<i>Secret Sharing Schemes</i>
<b>SCEP</b>	<i>Simple Certificate Enrollment Protocol</i>
<b>TLS</b>	<i>Transport Layer Security</i>
<b>TTP</b>	<i>Trusted Third Party</i>
<b>UML</b>	<i>Unified Modeling Language</i>
<b>VSSS</b>	<i>Visual Secret Sharing Schemes</i>
<b>VA</b>	<i>Validation Authority</i>
<b>WSDL</b>	<i>Web Service Definition Language</i>



# Glossary

**Abstract Syntax Notation One (ASN.1)** - is a standard and notation that describes rules and structures for representing, encoding, transmitting, and decoding data in telecommunications and computer networking.

**Authority Information Access (AIA)** - is a certificate extension which allows SSL/TLS clients (mostly web browsers) to go get the missing intermediary certificates, not presented by the server.

**Public Key Infrastructure (PKI)** - is a set of hardware, software, people, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates.

**Enterprise Java Beans Certificate Authority (EJBCA)** - is an enterprise class PKI Certificate Authority built on *Java Enterprise Edition (JEE)* technology.

**Certificate Authority (CA)** - A CA issues certificates to, and vouches for the authenticity of entities. The level of trust you can assign to a CA is individual, per CA, and depends on the CA's Policy (CP) and CA Practices Statement (CPS).

**Registration Authority (RA)** - An RA is an administrative function that registers entities in the PKI. The RA is trusted to identify and authenticate entities according to the CA's policy. There can be one or more RA's connected to each CA in the PKI.

**Root CA** - A RootCA has a self-signed certificate and is also called Trusted Root. Verification of other certificates in the PKI ends with the RootCA's self-signed certificate. Since the RootCA's certificate is self-signed it must somehow be configured as a trusted root for all clients in the PKI.

**Sub CA** - A subordinate CA, or SubCA for short, is a CA whose certificate is signed by another CA, that can be another SubCA or a RootCA. Since the SubCA's certificate is signed by another CA, it does not have to be configured as a trusted root. It is part of a certificate chain that ends in the RootCA.

**Validation Authority (VA)** - A VA is responsible for providing information on whether certificates are valid or not. There can be one or more VA's connected to each CA in the PKI.

**End-Entity** - An end-entity is a user, such as an e-mail client, a web server, a web browser or a VPN-gateway. End-entities are not allowed to issue certificates to other entities, they make up the leaf nodes in the PKI.



# Notation

## Sets

$X \subseteq Y$	$X$ is a subset of $Y$
$X \subset Y$	$X$ is a subset of $Y$ and $X \neq Y$
$\mathcal{P}(X)$	the powerset of $X$ , i.e., the set of all subsets of $X$
$\cup$	the set union
$\cap$	the set intersection
$\setminus$	the set difference
$\overline{X}$	the complement of set $X$ with respect to a given superset
$ X $	the cardinality of $X$
$\emptyset$	the empty set
$\mathbb{Z}$	denotes the set of integers
$\mathbb{Z}_p$	denotes the set of prime numbers
$\mathbb{N}$	denotes the set of positive integers (natural numbers)
$GF_q$	the Galois field of order $q$ , where $q$ is a prime power

## Secret Sharing

$n$	the total number of participants
$1, 2, \dots, n$	the (labels of) participants
$\mathcal{A}$	the authorized access structure ( $\mathcal{A} \subseteq \mathcal{P}(\{1, 2, \dots, n\})$ )
$\mathcal{A}_{min}$	the set of the minimal authorized groups
$\overline{\mathcal{A}}$	the unauthorized access structure (the complement of $\mathcal{A}$ )
$k$	the threshold (for the case of threshold secret sharing)
$\mathcal{S}$ (or $S_0$ )	the set of secrets
$S$	the secret
$U_i$	the user indexed to index $i$
$a_i$	the share corresponding to the user $i$
$\mathcal{TP}$	the trusted third party (or dealer)



# Chapter 1

## Introduction

A fundamental aspect of modern information society is security which has now become more than ever an asset of great importance in almost every area. Nowadays confidentiality and privacy of computational frameworks, along with high-level authentication, are considered essential assets of any system.

The distribution and management of secrets and cryptographic keys has historically always been a difficult problem. The process is even more complicated when a secret needs to be divided and shared amongst a group of entities and when group access, authorization and authentication at a given time is required. For instance in critical systems where all the players may not be simultaneously available and where an *access right* must be shared between a group of entities, redundancy must be built into the system and thus it is often necessary that only a subset of entities from this group simultaneously authenticate themselves in order to gain the *access right* or recover the *secret*.

The need for protocols which secure the distribution of tasks while protecting the privacy of users and the reliability of results lead to the development of cryptographic protocols to address these issues. These are denoted by literature as secure multi-party computation problems (Yao82). Secure multiparty computation enable entities as a group to compute some function over their inputs, while at the same time keeping these inputs private.

Our work focuses on two cryptographic systems, the *Secret Sharing Schemes* and *Public Key Infrastructure*.

*Secret Sharing Schemes* refers to methods for distributing a *secret* amongst a group of participants, each of whom is allocated a *share* (or *shadow*) of the secret. The recovery of the secret is “limited” by a *threshold* on the number of participants recovering the secret. The secret can only be reconstructed when a sufficient number (above or equal to the *threshold*) of shares are combined together.

*Public Key Infrastructure* reliably verifies the identity of a user via *digital signatures*, these are the *de facto* technique for safeguarding electronic communications and transactions. Just like passports, diplomas, identity cards, and other traditional certificates, they can specify any kind of data. Digital certificates are already widely used on the Internet, to authenticate e-mail, web servers, and software. The most popular web browsers have built-in capabilities for storing, sending, and verifying digital certificates.

Digital certificates are also playing an increasingly important role in electronic payments (“e-Commerce”, “e-Banking”), access control (to Web sites, databases, institutions, etc), cloud-computing, grid-computing, digital copyright protection, electronic voting, and so on.

Around the world, transport organizations, municipalities, health care providers, financial institutions, military departments and other influential organizations are using or are planning to provide their constituents with digital certificates with the sole means of participating in their systems. Already, digital certificates may be built into any device or piece of software that must be able to communicate securely with other devices or with individuals. This includes mobile phones,

watches, televisions, cars, and conceivably even computerized household appliances.

Digital certificates are traditionally associated with a single entity, which can be an individual or public or private corporation. A certificate used in a *Public Key Infrastructure* specifies a binding between the entities name and the entities public key. On the other hand it is often desired that the power to digitally sign, authenticate and authorize actions and events is shared amongst a group of actors. A real life example is where a relative of a patient and at least one doctor must concur in order to authorize possible dangerous medical procedures. A *Threshold Crypto-System* is a system where it is necessary that a number bodies larger than a “threshold” value cooperate during the decryption protocol.

In sensitive and critical environments in order to increase *trust* in the system and improve confidentiality and for sheer necessity sensitive keys can be broken up and stored at several physical locations or as, we shall demonstrate in this, thesis on *digital certificates*.

## 1.1 Motivation and Objectives

The objectives of this thesis are to explore the possibility of using *Secret Sharing Schemes* for use in *digital certificates*. This thesis will design, develop an implementation of a *proof of concept* prototype of a cryptographic framework that uses *Threshold Cryptography* methodologies supported by *Public Key Infrastructure*. The support for a *Threshold Cryptography Scheme* is embedded in the digital certificate in a form of a custom certificate extension, as a *security token*.

An objective of the framework technology is that it should also be built on technologies that support scalability and redundancy for deployment in a cloud environment, as nowadays almost all business models request, or require, such features in their products/technologies.

A final objective is a security analysis of the architecture and implementation to be developed.

One of the motivations for this thesis was the integration of this work in the project PRICE - *Privacy, Reliability and Integrity in Cloud Environments*. In collaboration with PT *Inovação* in the context of the *Plano de Inovação* 2012-2014.

## 1.2 Contributions

The main contribution of this thesis is an attempt to solve issues, as is referred in the literature, in multi-party computation problems.

It enables users to perform group based decisions based on previously “agreed” *thresholds* which takes into account the privacy and confidentiality of the entities involved. Hence allowing users to better manage their “trust” in a group, or with other users.

It addresses the key distribution problem as the keys are managed and distributed by the CA and also deals with hierarchy as its supported by a PKI, which is inherently hierarchical, hence the different privileges of a user associated with a certificate can be dealt with by the CA.

Also, whilst developing this thesis and studying other collaborative mechanisms to implement frameworks for *Secret Sharing Schemes*, other than digital certificates, a framework using smart-

cards was developed and this work published in the proceedings of the European Conference on Information Warfare and Security, 2013 (CP13).

## 1.3 Thesis Outline

The remaining of this thesis is structured as follows:

- Chapter 2: Presents the state-of-the-art with constructions of *Shamir's Secret Sharing Scheme* and *Blakley's Secret Sharing Scheme*. Also there's an overview on *Secret Sharing Schemes* and the evolution of PKI and Digital Certificate Extensions;
- Chapter 3: Introduces some considerations on cryptographic premisses. Outlines requirements and features on some of the technologies used to implement our framework;
- Chapter 4: Describes the architecture of our framework;
- Chapter 5: Describes methodologies used to implement our framework;
- Chapter 6: Contains some security analysis of framework and discussions;
- Chapter 7: Discussions, conclusions and future work.





# Chapter 2

## State of the Art

### Contents

1.1	Motivation and Objectives . . . . .	2
1.2	Contributions . . . . .	2
1.3	Thesis Outline . . . . .	3

## 2.1 Secret Sharing Schemes

This chapter introduces some basic notions and theory concerning *Secret Sharing Schemes* (SSS). There has been a considerable amount of research on SSS since their seminal, independent, announcement by Shamir and Blakley (Sha79) (Bla79) in 1979 and this chapter will therefore review some of this research, the various methodologies, implementations, breakthroughs, etc. We will focus primarily on Shamir's constructions as our work will make use of Shamir's scheme.

A SSS begins with a *secret* and derives from it certain *shares* (or *shadows*) which are distributed to the participants. This *secret* may be recovered only by a certain predetermined groups of participants. The set of all groups, or “qualified subset”, which can reconstruct the *secret* is referred to in the literature as *access structure*.

The reconstruction of the secret can be made by the participants after they pool together their shares or by a special party, called *combiner*, after receiving the shares from the participants of an *authorized group*.

Usually, a SSS is coordinated by a *dealer* who, of course, has to be a *mutually trusted party*, but there are also SSS which can be configured without the presence of a *dealer* (ZLL11).

## 2.2 Threshold Secret Sharing Scheme

### 2.2.1 Shamir's Secret Sharing Scheme

In this section we will demonstrate the construction of the SSS as enunciated by Shamir in (Sha79). This is the construction we are going to use in our work.

Shamir's scheme (Sha79) is based on polynomial interpolation and in the fact that: any given  $k$  pairs in a 2-dimensional plane  $(x_1, y_1), \dots, (x_k, y_k)$  with  $x_i \neq x_j$  for all  $1 \leq i < j \leq k$ , there's one and only one polynomial  $Q(x)$  of degree  $k - 1$  such that  $Q(x_i) = y_i$ , for all  $1 \leq i \leq k$ .

## Distribution of Secret Shares

A *Trusted third-party*  $\mathcal{TTP}$  (or dealer) distributes shares of a secret  $S$  to  $n$  users.

1. The  $\mathcal{TTP}$  sets the secret  $S$  as the free coefficient of a random polynomial  $Q$  of degree  $k - 1$  over the field of the positive integers modulo a large prime  $p$ , where  $p > n \wedge p > S$ , and defines  $a_0 = S$ ;
2. The  $\mathcal{TTP}$  selects  $k-1$  random, independent coefficients  $a_1, \dots, a_{k-1}$ , such that  $0 \leq a_j \leq p-1$ , defining the random polynomial over  $\mathbb{Z}_p$ ,  $Q(x) = \sum_{j=0}^{k-1} a_j x^j$ ;
3. The  $\mathcal{TTP}$  computes  $S_i = Q(i) \bmod p$ ,  $1 < i \leq n$  (or for any distinct points  $i$ ,  $1 \leq i \leq p-1$ ), and securely transfers the share  $S_i$  to user  $U_i$ , along with public index  $i$ .

## Recovery of Secret

Any group of  $k$  or more users pool their shares. Their shares provide  $k$  distinct points  $(x, y) = (i, S_i)$  allowing computation of the coefficients  $a_j$ ,  $1 \leq j \leq k - 1$  of  $Q(x)$  by Lagrange interpolation (see equations bellow). The secret is recovered by as  $f(0) = a_0 = S$ .

The coefficients of an unknown polynomial  $Q(x)$  of degree less than  $k$ , defined by points  $(x_i, y_i)$ ,  $1 \leq i \leq k$ , are given by the Lagrange interpolation formula:

$$Q(x) = \sum_{i=1}^k y_i \prod_{j=1, j \neq i}^k \frac{x - x_j}{x_i - x_j} \bmod(p)$$

$$S = Q(0) = \sum_{i=1}^k y_i \prod_{j=1, j \neq i}^k \frac{-x_j}{x_i - x_j} \bmod(p)$$

## Example of Shamir's Secret Sharing Scheme

Next we will provide a example of secret distribution/recovery (with artificially small parameters) for a better understanding of Shamir's SSS.

Let  $n = 5$  and  $k = 3$ . Let us consider polynomial  $Q(x) = 5x^2 + 3x + 7$  over the field  $\mathbb{Z}_{11}$ .

We get following *shares* of the *secret*.

$$\begin{aligned}
\text{Secret} &= Q(0) = 7 \\
U_1 &= 5 \times 1^2 + 3 \times 1 + 7 \bmod 11 = 4 \\
U_2 &= 5 \times 2^2 + 3 \times 2 + 7 \bmod 11 = 0 \\
U_3 &= 5 \times 3^2 + 3 \times 3 + 7 \bmod 11 = 6 \\
U_4 &= 5 \times 4^2 + 3 \times 4 + 7 \bmod 11 = 0 \\
U_5 &= 5 \times 5^2 + 3 \times 5 + 7 \bmod 11 = 4
\end{aligned}$$

Considering a poll of shares  $U_1, U_2$  and  $U_5$  the secret can be reconstructed as:

$$\begin{aligned}
4 \times \frac{2}{2-1} \times \frac{5}{5-1} + 0 \times \frac{1}{1-2} \times \frac{5}{5-2} + 4 \times \frac{1}{1-5} \times \frac{2}{2-5} &= 7 \\
7 &\equiv 7 \text{ q.e.d.}
\end{aligned}$$

### 2.2.2 Blakley Secret Sharing Scheme

In Blakley's scheme, as presented in (Bla79), the secret is an element of the vector space  $GF_q^k$ . The shares are any  $n$  distinct  $(k-1)$ -dimensional hyperplanes that contain the secret, where an  $(k-1)$ -dimensional hyperplane is a set of form:

$$(x_1, \dots, x_k) \in GF_q^k \mid \alpha_1.x_1 + \dots + \alpha_k.x_k = \beta$$

Where  $\alpha_1, \dots, \alpha_k, \beta$  are arbitrary elements of the field  $GF_q$ . The secret can be obtained by intersecting any  $k$  shares.

In figure 2.1 we pictorially describe the case  $k = 2$ .

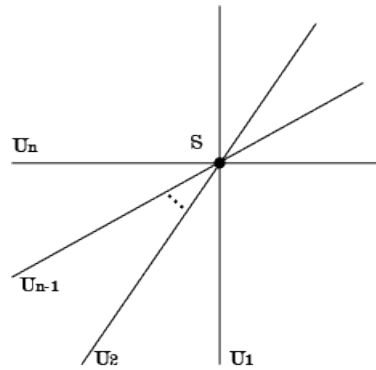


Figure 2.1: Blakley Secret Sharing Scheme for threshold  $k = 2$ .

Blakley's scheme isn't *perfect* because any unauthorized group knows that the secret lies in the intersection of their hyperplanes. Nevertheless, this scheme can be modified to achieve *perfect* security by choosing the secret as a single coordinate of a point in  $GF_q^k$ , with the cost of affecting the information rate.

## 2.3 Access Structures

When discussing SSS it is often useful to consider their properties in relation to sets and information theory properties.

The elements of the *access structure* will be referred to as the *authorized* groups/sets and the rest are called *unauthorized* groups/sets.

As Ito, Saito, and Nishizeki have remarked in (ISN87), any *access structure*  $\mathcal{A}$  must satisfy the following natural condition <sup>1</sup>.

$$(\forall B \in \mathcal{P}(\{1, 2, \dots, n\}))((\exists A \in \mathcal{A})(A \subseteq B) \Rightarrow B \in \mathcal{A})$$

Here  $\mathcal{P}$  denotes the power set of participants labelled  $(\{1, 2, \dots, n\})$ . This condition basically states that if a group can recover the secret, so can a larger group. Benaloh and Leichter denoted such access structures monotone in (BL90). From this we can also derive the property for the unauthorized access structure  $\overline{\mathcal{A}}$ :

$$(\forall B \in \mathcal{P}(\{1, 2, \dots, n\}))((\exists A \in \overline{\mathcal{A}})(B \subseteq A) \Rightarrow B \in \overline{\mathcal{A}})$$

This means that if a group cannot recover the secret, neither can a smaller group.

Any monotone authorized access structure  $\mathcal{A}$  is well specified by the set of the minimal authorized groups, i.e., the set:

$$\mathcal{A}_{min} = \{A \in \mathcal{A} \mid (\forall B \in \mathcal{A} \setminus A)(\neg (B \subseteq A))\}$$

We will use the notation  $\mathcal{A} = cl(\mathcal{A}_{min})$  and  $\mathcal{A}_{min}$  will be referred to as the *basis* of  $\mathcal{A}$ . In general the *closure* of some  $\mathcal{C} \subseteq \mathcal{P}(\{1, 2, \dots, n\})$ , denoted by  $cl(\mathcal{C})$  is defined as:

$$cl(\mathcal{C}) = \{A \in \mathcal{P}(\{1, 2, \dots, n\}) \mid (\exists C \in \mathcal{C})(C \subseteq A)\}$$

The unauthorized access structure  $\overline{\mathcal{A}}$  is well specified by the set of the maximum authorized groups, i.e., the set:

$$\overline{\mathcal{A}}_{max} = \{A \in \overline{\mathcal{A}} \mid (\forall B \in \overline{\mathcal{A}} \setminus A)(\neg (A \subseteq B))\}$$

**Example 2.1.** Let us consider  $n = 4$  and the access structure  $\mathcal{A} = \{\{1, 2\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 2, 3, 4\}, \{3, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}$ . We obtain that  $\mathcal{A}_{min} = \{\{1, 2\}, \{3, 4\}\}$ ,  $\overline{\mathcal{A}} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}\}$  and  $\overline{\mathcal{A}}_{max} = \{\{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}\}$

<sup>1</sup>There are papers (see, for example, (Beu89) or (BSGV93)) that consider non-monotone access structures. More precisely, in these schemes, there are positive and negative shares which lead to veto capabilities. As Obana and Kurosawa have remarked in (OK96), this feature is possible only in the case that we assume that the reconstruction process is trustworthy. The simplest solution for the veto feature is that the opposing participants possess a special “veto” share in the secret reconstruction phase, leading to an incorrect secret.

Depending on the “quantity” of the secret-information leaked to an unauthorized group, we have the following categories:

- Perfect Secret Sharing Schemes - The shares of any unauthorized group give no information (in Information-Theoretic sense) about the secret.
- Computationally Secure Secret Sharing Schemes - Some information about the secret is leaked to the unauthorized groups, but the problem of finding the secret is intractable <sup>2 3</sup>.

Under the scope of *Theory of Information* we can say, if the length of every share is the same as the length of the secret then these schemes are *ideal* (KGH83). Shamir’s SSS is such an example.

## 2.4 Overview of Secret Sharing Schemes

In the initial stages of work on secret sharing, Blakley (Bla79) and Shamir (Sha79) considered only schemes with a  $(k, n)$  - threshold access structure. Benaloh showed an interactive verifiable  $(k, n)$  - threshold SSS which is zero knowledge (Ben87).

In (BS97), D. R. Stinson and C. Blundo introduced the anonymous threshold scheme. Informally, in an anonymous SSS, the secret is reconstructed without the knowledge of, which participants hold which shares. In such schemes the computation of the secret can be carried out by giving the shares to a “black box” that doesn’t know the identities of the participants holding those shares. The authors proved a lower bound on the size of the shares for anonymous threshold schemes and provided optimal schemes for certain classes of threshold structures by using a combinatorial characterization of optimal schemes. Further results can be found in (CS90).

The original SSS by Shamir (Sha79) and Blakley (Bla79) are  $(k, n)$  - threshold SSS. But, there exist some extensions, which are shown in table 2.1 and summarized as follows.

Extensions of $(k, n)$ - Threshold Secret Sharing Schemes		
	<i>Threshold Type</i>	<i>General Access Structure</i>
<b>Perfect SS Schemes</b>	Shamir (Sha79), Blakley (Bla79)	Ito et al. (ISN87)
<b>Ramp SS Schemes</b>	Blakley-Meadows (BM85), Yamamoto (Yam86)	Kurosawa et al. (KOS <sup>+</sup> 93)

Table 2.1: Extensions of  $(k, n)$  - Threshold Secret Sharing Schemes

In the case of  $(k, n)$  - threshold access structure, we assume that every share is equally important, but there are cases we want to make some shares more important than the others, or *hierarchize* the shares.

A recommended read on the topic of *Ideal Hierarchical Secret Sharing Schemes* is (Far10).

One can think in many situations in which, some participants should be more powerful than others. A first realization to overcome this limitation on initial SSS was made by Shamir in his seminal work (Sha79) by proposing a simple modification of the threshold scheme to be used by organizations. Namely, every participant receives as its share a certain number of shares from a threshold scheme, according to its position in the hierarchy.

<sup>2</sup>A problem is called computationally intractable if there is no polynomial deterministic algorithm for solving it in a feasible time frame.

<sup>3</sup>Ramp Schemes, Secret Sharing Schemes based on information dispersal or Secret Sharing Schemes based on public information are examples of computationally secure schemes. Such schemes can lead to shorter shares.

In this way a scheme for a *weighted threshold access structure* is obtained. That is, every participant has a *weight* (a positive integer) and a set is qualified if and only if its *weight sum* is at least a given threshold.

This scheme is not *ideal* because the shares have in general larger length than the secret.

In order to realize a secure data storage efficiency, we must make a size of each share as small as possible, and hence, efficient coding methods for SSS with general access structures must be established. It's established how to construct efficient  $(k, n)$  - threshold in SSS (Sha79).

As an extension of  $(k, n)$  - threshold SSS, Ramp SSS were proposed independently by Blakley-Meadows (BM85) and Yamamoto (Yam86).

A Ramp SSS is a SSS with intermediate properties between qualified sets and forbidden sets. Ramp SSS (BM85), (Yam86) are threshold SSS denominated  $(k, L, n)$  - threshold Ramp SSS.

The  $(k, L, n)$  threshold Ramp SSS are designed such that a secret  $S$  can be decrypted from arbitrary  $k - out - of - n$  shares but no information of  $S$  can be obtained from arbitrary  $k - L$  or less shares. Considering the case of  $L = 1$ ,  $(k, L, n)$  - threshold Ramp SSS  $L$  falls back to a  $(k, n)$  - threshold SSS, and hence, Ramp SSS can be considered as an extension of  $(k, n)$  - threshold SSS.

Yvo Desmedt coined the term *Threshold Cryptography* in (Des87). He describes *Threshold Cryptography* as a society-oriented cryptography.

In *Threshold Cryptography*, the message receiver is not an ordinary individual, but an entity of an organization, such as a small group of people or an entire department. These users share responsibilities of the entity to decrypt a message or sign a message, which is a desirable way to prevent power abuse, as mentioned by Y. Desmedt (DF90).

Also, in *Threshold Cryptography* each entity owns a public key and the corresponding private key is shared among a group of, say  $n$ , users. Any  $b$ ,  $1 \leq b \leq n$ , of these  $n$  users can co-decrypt or co-sign a message without reconstructing the shared private key. On the other hand, any subset with size less than  $t$ ,  $1 \leq t \leq n$ , users can neither recover the private key or co-sign/co-decrypt a message on behalf of the entity (Des98).

So far several threshold RSA primitives have been presented in the research community, not only by Y. Desmedt (DCB94) (DF94) but also by Y. Frankel (FD92), R. Gennaro (GRJK07) and T. Rabin (Rab98) as well as in threshold DSA primitives, by R. Gennaro (GJKR99) and S.K. Langford (Lan95).

Benaloh describes an homomorphism property that is present in many threshold schemes which allows of multiple secrets to be combined to form composite shares which are shares as a composition of the secrets (Ben96) (Ben87).

The application of secret sharing homomorphism includes fault-tolerant verifiable secret-ballot elections as well as verifiable secret sharing, this is especially useful in the case of *e-voting systems*.

In the approach on homomorphism used by Feldman (Fel87) the secret is distributed in such a way as to enable each trustee to, without further interaction, verify that its share is a well-formed and valid share of the secret.

According to João Mendes (MM11) *homomorphic* is an adjective that describes a property of an encryption scheme that has the ability to perform computations on the cipher-text without decrypting it first.

In homomorphic threshold crypto-systems a specific algebraic operation is performed on the plain-text and another (possibly different) algebraic operation is performed on the cipher-text. Depending on one's viewpoint, this can be seen as either a positive or negative attribute of the crypto-system.

Homomorphic encryption schemes are malleable by design. The homomorphic property of various crypto-systems can be used to create secure voting systems, collision-resistant hash functions, and private information retrieval schemes.

As mentioned earlier, SSS have amounted extensive research throughout out the years, the following table 2.2 summarizes several different realizations of SSS from ordinal SSS to *Quantum Secret Sharing Schemes* (QSSS).

Various implementations of Secret Sharing Schemes			
<i>Based on</i>	<i>Designation</i>	<i>Secret Information</i>	<i>Proposed by</i>
<b>Computers</b>	Threshold SSS <sup>4</sup>	Numbers in finite fields	Shamir (Sha79), Blakley (Bla79)
<b>Human sense</b>	Visual Cryptography	Images	Naor-Shamir (NS94)
	Cerebral Cryptography	3D Images	Desmedt et al.(DHQ98)
	Optical Cryptography	Light	Desmedt et al.(YDQ98)
	Audio Cryptography	Sound	Desmedt et al.(YDQ98)
	Tempo-based audio Cryptography	Rhythm	Chiou-Laih (CL03)
<b>Quantum Information</b>	Quantum SSS Quantum SSS	Numbers Quantum states	Hillery et al. (HBB99) Cleve et al. (CGL99)

Table 2.2: Various implementations of Secret Sharing Schemes

The audio cryptography (CL03), (YDQ98), the optical cryptography, (YDQ98) and the cerebral cryptography (YDQ98) are SSS which use human senses in decryption in the same way as *Visual Secret Sharing Schemes* (VSSS).

In the audio and optical cryptography (YDQ98), the secret and shares are sounds or lights which can be considered as waves, and the interference of waves is used in decryption. In other words, the waves of shares corresponding to a qualified set are mutually strengthened to “listen” to or “see” the secret, but the waves of shares for the forbidden set are mutually weakened to hide the secret.

The audio cryptography (YDQ98) is not unconditionally secure, although the tempo-based audio cryptography proposed in (CL03) can guarantee unconditional security. In the tempo-based audio cryptography, secret bits are encrypted into rhythms, and security assumptions are similar to VSSS.

The cerebral cryptography is a SSS based on the so-called stereogram. The stereogram (Jul60) is an eye-sight illusion that perceives a 3-dimensional image from two 2-dimensional images. However, the security conditions are not clarified in (DHQ98).

More recently, quantum cryptography has been extensively studied. As in the case of classical cryptography, quantum cryptography is also designed for secure data transmissions or secure data storage.

The first quantum cryptography for data transmission is the so-called *BB84 protocol* proposed by Bennett-Brassard in 1984 (BB84), which is a key distribution protocol. On the other hand, for secure data storage, QSSS are proposed in (CGL99), (Got99), (HBB99), (KKI99).

<sup>4</sup>These are sometimes denominated in nowadays literature as part of *Threshold Cryptography* or *Threshold Crypto-Systems* when related to PKI and where initially called as such by Yvo Desmedt in (Des87).

Compared with classical cryptography, quantum cryptography has remarkable advantages such that it can detect an eavesdropper and a dishonest participant by measurements of quantum states.

The first QSSS (HBB99) is a three-party protocol based on three entangled particles entitled Greenberger-Horne-Zeilinger (GHZ) state. In this QSSS, the measurement result for one share can be determined by combining measurement results for the other two shares. Hence, this method in (HBB99) can be considered as an extension of a QSSS rather than a QSSS. A  $(k, n)$  - threshold QSSS is considered in (KKI99) as an extension of the method in (HBB99).

In QSSS treated in (HBB99), (KKI99), secret information is ordinary bits which are encoded into quantum states. On the other hand, its proposed in (CGL99), (Got99) to encrypt a secret quantum state into shares.

Its shown in (CGL99) that  $(k, n)$  - threshold QSSS can be realized only in the case that  $n \leq 2k - 1$ , which comes from the requirement of the so-called *no-cloning theorem*. Furthermore, in the case that a secret quantum state is a pure-state, it must hold that  $n = 2k - 1$ . It is also shown in (Got99) that QSSS for general access structures can be constructed for any mixed-state secret quantum states if these access structures satisfy the *no-cloning theorem*. The coding efficiency of QSSS is also treated in (Got99), (OTH03).

SSS, VSSS, and QSSS can guarantee unconditional security, but computationally secure SSS are considered in (Cac95), (Kra93). In the case of computationally secure SSS, the coding rates of shares are much more efficient than unconditionally secure SSS (Kra93). Furthermore, such SSS can treat plural secrets dynamically without redistributing new shares to participants secretly (Cac95).

## 2.5 Evolution of Public-Key Infrastructure

Individuals and organizations often have a legitimate need to verify the identity or other attributes of the individuals and entities that they communicate or transact with as well as communicate with these entities in a secure and confidential fashion.

The development of public-key cryptography is one of greatest revolutions in the history of cryptography. From its earliest beginnings to modern times, virtually all cryptographic systems have been based on the elementary tools of substitution and permutation, however public-key cryptography provides a radical departure from previous cryptographic approaches. For one thing, public-key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is *asymmetric*, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

In many applications, symmetric cryptographic techniques are inappropriate: they require a *trusted third party* to set up a secret key for any two parties that have not communicated previously, and cannot offer non-repudiation. Thus, there is a fundamental need for public-key cryptography.

The concept of public-key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption.

The first problem is that of key distribution. The second problem that Diffie and Hellman pondered, and one that was apparently unrelated to the first, was that of *digital signatures*.



If the use of cryptography was to become widespread, not just in military situations but for commercial and private purposes, then electronic messages and documents would need the equivalent of signatures used in paper documents. That is, could a method be devised that would stipulate, to the satisfaction of all parties, that a digital message had been sent by a particular person?

Diffie and Hellman achieved an astounding breakthrough in 1976 (DH76) by coming up with a method that addressed both problems and was radically different from all previous approaches to cryptography <sup>5</sup>.

Public-key cryptography enables the parties in a system to digitally sign and encrypt their messages. When two parties that have not communicated before want to establish an authenticated session, they need merely fetch the public key of the other, there is no need for a trusted third party to mediate every transaction.

In their seminal paper (DH76) on public-key cryptography, Diffie and Hellman also pointed out the problem of authenticating that a public key belongs to an entity. They suggested using secure online repositories with entries that specify name-key bindings. In 1978, Kohnfelder (Koh78) proposed to avoid this potential bottleneck by having a trusted entity, called the CA <sup>6</sup>, vouch for the binding between a public key and its holder.

A *digital certificate* is a signed assertion about a public key. More specifically, it is a *digital signature* of the CA that binds a public key to some other piece of information. This enables all system participants to verify the name-key binding of any presented certificate by applying the public key of the CA.

A *public key infrastructure* (PKI), is an infrastructure for a distributed environment that centers around the distribution and management of public keys and digital certificates. It is widely recognized that PKIs are an essential ingredient for secure electronic communications and transactions in open environments.

The CA can be made responsible not only for certifying public keys and authenticating certificate applicants, but also for notarizing electronic documents, resolving disputes, and keeping track of revoked keys.

Some or all of these functions may be managed by separate trusted parties. For instance, the registration and approval of certificate applicants may be done by a separate *Registration Authority* (RA).

In practice, a PKI can have multiple CA's, so that certificate applicants and verifiers need not trust a single CA. CA's can certify the public keys of other CA's, and in this manner arbitrary CA structures can be formed. This gives rise to such notions as certificate chains, bridge CA's, and cross certification. These techniques enable anyone to be the issuer of their own digital certificates, and all issuers can coexist in a single PKI.

---

<sup>5</sup>Diffie and Hellman first publicly introduced the concepts of public-key cryptography in 1976. Hellman credits Merkle with independently discovering the concept at that same time, although Merkle did not publish until 1978 (Mer78). In fact, the first unclassified document describing public-key distribution and public-key cryptography was a 1974 project proposal by Merkle (<http://merkle.com/1974>). However, this is not the true beginning. Admiral Bobby Inman, while director of the National Security Agency (NSA), claimed that public-key cryptography had been discovered at NSA in the mid-1960s (Sim93). The first documented introduction of these concepts came in 1970, from the Communications-Electronics Security Group, Britain's counterpart to NSA, in a classified report by James Ellis (Ell70). Ellis referred to the technique as non-secret encryption and describes the discovery in (Ell99)

<sup>6</sup>In recent years the term *Trusted Third Party* (TTP) has gained in popularity.

The X.509 certificate framework (Uni97) is the best known example of *identity certificates*. In 1988, the International Telecommunications Union (ITU) started working on X.509. X.509v1 was designed to certify the public keys of principals that are uniquely named in X.500 (CCI88) (GR97) (Wah97), an online database listing globally unique names. An entry in an X.500 directory can be a person, a device, or anything else that can be assigned a “Distinguished Name”. X.509v2, released in 1993, provided for a more flexible choice of identifiers.

### 2.5.1 Digital Certificate Extensions

The third version of the X.509 certificate framework, X.509v3, released in June 1997, greatly improved the flexibility of X.509 certificates, by providing for a generic mechanism to extend certificates. This previous statement sets the ground for the development of our work, it unfolds the plausibility of certificate extension to support *Threshold Cryptography*.

Also, several pilot PKI projects conducted by U.S. federal agencies (including the NSA, the IRS, the FBI, the U.S. Department of Defense, and the Social Security Administration) as part of the Federal Public Key Infrastructure (Com98) (FPKI) use X.509v3 certificates, with application-dependent extensions.

## 2.6 Related Work

Already, one can find several Web applications, libraries or simple “recreational” applications of Shamir’s SSS online these normally consist of a desktop or web application where a single user enters a secret and then the dealer distributes all the shares to the user who is responsible for there storage and distribution. An example of such software can be found for example at <http://point-at-infinity.org/ssss/>. Another related software is Nightingale from RSA Security which uses secret splitting to secure sensitive data (Bra) by dividing a user’s password (or other key) into shares for two independent servers, the scheme is used in password authentication. Another example is Finnegan Lab System, a graphical desktop application designed for a single user to learn and explore how secret sharing works (Ols04). However there seems to be no practical implementation of a collaborative mechanism to share secrets between a group of mutually authenticated users.

# Chapter 3

## Requirements and Features

### Contents

<b>2.1</b>	<b>Secret Sharing Schemes . . . . .</b>	<b>5</b>
<b>2.2</b>	<b>Threshold Secret Sharing Scheme . . . . .</b>	<b>5</b>
2.2.1	Shamir's Secret Sharing Scheme . . . . .	5
2.2.2	Blakley Secret Sharing Scheme . . . . .	7
<b>2.3</b>	<b>Access Structures . . . . .</b>	<b>8</b>
<b>2.4</b>	<b>Overview of Secret Sharing Schemes . . . . .</b>	<b>9</b>
<b>2.5</b>	<b>Evolution of Public-Key Infrastructure . . . . .</b>	<b>12</b>
2.5.1	Digital Certificate Extensions . . . . .	14
<b>2.6</b>	<b>Related Work . . . . .</b>	<b>14</b>

In this section we assess and specify the necessary requirements for the implementation of the framework, as well as some features of the selected PKI CA chosen.

We also denote some preliminary reviews on cryptographic subjects that should be considered.

## 3.1 Preliminaries

### 3.1.1 Object Identifier (OID)

An *Object Identifier* (OID) is an identifier used to name an object (as implied). Structurally, an OID consists of a node in a hierarchically-assigned namespace, formally defined using the ITU-T's ASN.1 standard, X.690. Successive numbers of the nodes, starting at the root of the tree, identify each node in the tree.

Users set up new nodes by registering them under the node's registration authority. The root of the tree contains the following three arcs:

0 : ITU-T

1 : ISO

2 : joint-iso-itu-t

All other arcs branch from these 3, as shown of figure 3.1.

These OIDs are used extensively in computer security. OIDs serve to name almost every object type in X.509 certificates e.g. Distinguished Names, Certificate Policy Statements, Certificate extensions, etc.

### 3.1.2 Abstract Syntax Notation One (ASN.1)

*Abstract Syntax Notation One* (ASN.1) is a standard and notation that describes rules and structures for representing, encoding, transmitting, and decoding data in telecommunications and com-

## Top of the OID tree

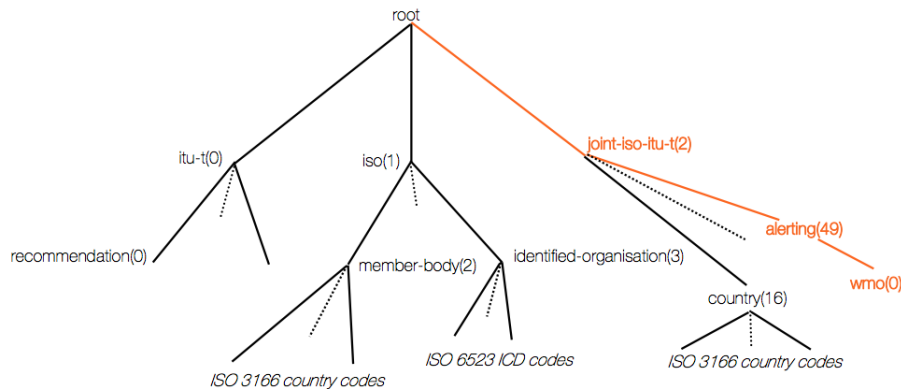


Figure 3.1: OID tree

puter networking. The formal rules enable representation of objects that are independent of machine-specific encoding techniques. Formal notation makes it possible to automate the task of validating whether a specific instance of data representation abides by the specifications.

ASN.1 is a joint standard of the International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), and International Telecommunication Union Telecommunication Standardization Sector ITU-T, originally defined in 1984 as part of CCITT X.409:1984. ASN.1 moved to its own standard, X.208, in 1988. The revised 1995 version is covered by the X.680 series. The latest available version is dated 2008, and is backward compatible with the 1995 version.

ASN.1 has sets of rules to precisely specifying how messages must be “encoded” for communication with other machines. Each set of “encoding rules” has specific characteristics, such as compactness or decoding speed, which make it best suited for particular environments.

- *Basic Encoding Rules* (BER)
- *Canonical Encoding Rules* (CER)
- *Distinguished Encoding Rules* (DER) <sup>1</sup>
- *Packed Encoding Rules* (PER)

Applying an encoding rule to the data structures described by an abstract syntax provides a transfer syntax that governs how bytes in a stream are organized when sent between computers. The transfer syntax used by Distinguished Encoding Rules always follows a *Tag*, *Length*, *Value* format. The format is usually referred to as a TLV triplet in which each field (T, L, or V) contains one or more bytes, as shown on figure 3.2.



Figure 3.2: Tag, Length, Value triplet element.

The *Tag* field specifies the type of the data structure being sent, the *Length* field specifies the number of bytes of content being transferred, and the *Value* field contains the content. Note that the Value field can be a triplet if it contains a constructed data type as shown by the following figure 3.3.

<sup>1</sup>We will focus on this as it was created to satisfy the requirements of the X.509 specification for secure data transfer.

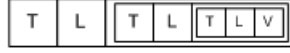


Figure 3.3: Tag, Length, Value triplet of a triplet.

The Length field in a TLV triplet identifies the number of bytes encoded in the Value field. The Value field contains the content being sent between computers. If the Value field contains fewer than 128 bytes, the Length field requires only one byte. Bit 7 of the Length field is zero (0) and the remaining bits identify the number of bytes of content being sent. If the Value field contains more than 127 bytes, bit 7 of the Length field is one (1) and the remaining bits identify the number of bytes needed to contain the length. Examples are shown in the following figure 3.4.

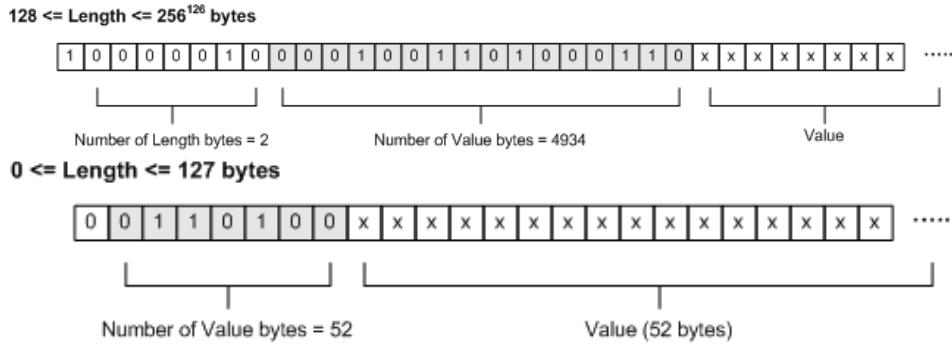


Figure 3.4: Examples of encoded length and value bytes

### 3.1.3 One-Way Functions

One-way functions are functions that are efficiently computable but infeasible to invert (in an average-case sense). That is, a function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is called one-way if there is an efficient algorithm that on input  $x$  outputs  $f(x)$ , whereas any feasible algorithm that tries to find a preimage of  $f(x)$  under  $f$  may succeed only with negligible probability (where the probability is taken uniformly over the choices of  $x$  and the algorithm's coin tosses).

Associating feasible computations with probabilistic polynomial-time algorithms, we obtain the following definition (Gol05).

**Definition 1.** A function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is called **one-way** if the following two conditions hold:

1. **Easy to evaluate:** there exists a polynomial-time algorithm  $A$  such that  $A(x) = f(x)$  for every  $x \in \{0, 1\}^*$
2. **hard to invert:** for every probabilistic polynomial-time algorithm  $A'$ , every polynomial  $p$ , and all sufficiently large  $n$ ,

$$Pr[A'(f(x), 1^n) \in f^{-1}(f(x))] < \frac{1}{p(n)}$$

where the probability is taken uniformly over all the possible  $x \in \{0, 1\}^n$  and all possible outcomes of the internal coin tosses of algorithm  $A'$ .

### 3.1.4 Pseudo Random Number Generators

An important cryptographic function is cryptographically strong pseudorandom number generation. *Pseudo Random Number Generators* (PRNG) are used in a variety of cryptographic and security applications. These applications give rise to two distinct and not necessarily compatible requirements for a sequence of random numbers: randomness and unpredictability.

- Randomness: commonly, the concern in the generation of a sequence of allegedly random numbers has been that the sequence of numbers be random in some well defined statistical sense. The following two criteria are used to validate that a sequence of numbers is random:
  - Uniform distribution: the distribution of bits in the sequence should be uniform. That is, the frequency of occurrence of ones and zeros should be approximately equal;
  - Independence: no one subsequence in the sequence can be inferred from the others; <sup>2</sup>.
- Unpredictability: the requirement is not just that the sequence of numbers be statistically random but that the successive members of the sequence are unpredictable. With “true” random sequences, each number is statistically independent of other numbers in the sequence and therefore unpredictable;
  - Forward unpredictability: if the seed is unknown, the next output bit in the sequence should be unpredictable in spite of any knowledge of previous bits in the sequence;
  - Backward unpredictability: it should also not be feasible to determine the seed from knowledge of any generated values. No correlation between a seed and any value generated from that seed should be evident; each element of the sequence should appear to be the outcome of an independent random event whose probability is 1/2.

In terms of *randomness*, the requirement for a PRNG is that the generated bit stream appear random even though it is deterministic. There is no single test that can determine if a PRNG generates numbers that have the characteristic of randomness. The best that can be done is to apply a sequence of tests to the PRNG. If the PRNG exhibits *randomness* on the basis of multiple tests, then it can be assumed to satisfy the *randomness* requirement.

---

<sup>2</sup>Although there are well defined tests for determining that a sequence of bits matches a particular distribution, such as the uniform distribution, there is no such test to “prove” independence. Rather, a number of tests can be applied to demonstrate if a sequence does not exhibit independence. The general strategy is to apply a number of such tests until the confidence that independence exists is sufficiently strong.

NIST SP 800-22 (nis) (A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications) specifies that the tests should seek to establish the following three characteristics.

- **Uniformity:** at any point in the generation of a sequence of random or pseudorandom bits, the occurrence of a zero or one is equally likely, i.e., the probability of each is exactly  $1/2$ . The expected number of zeros (or ones) is  $n/2$ , where  $n$  = the sequence length;
- **Scalability:** any test applicable to a sequence can also be applied to subsequences extracted at random. If a sequence is random, then any such extracted subsequence should also be random. Hence, any extracted subsequence should pass any test for randomness;
- **Consistency:** the behavior of a generator must be consistent across starting values (seeds). It is inadequate to test a PRNG based on the output from a single seed or an TRNG on the basis of an output produced from a single physical output.

Figure 3.5 shows two different forms of PRNGs, based on application.

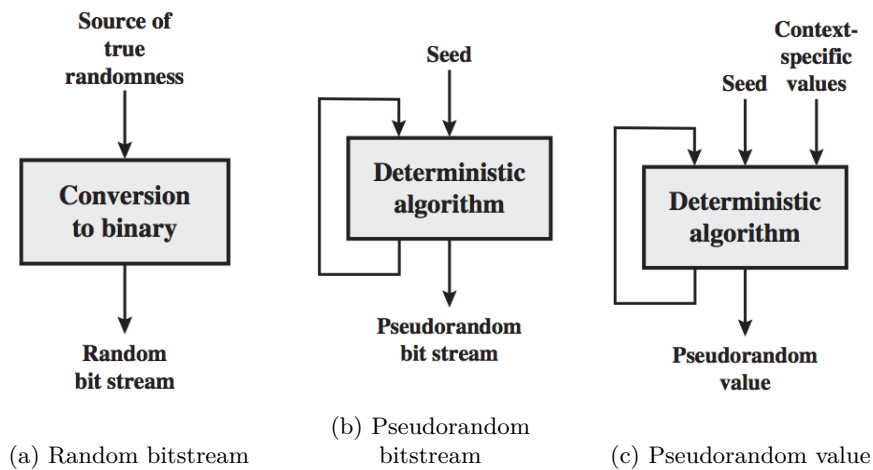


Figure 3.5: Random and Pseudorandom Number Generators (Sta11)

All passwords used our framework (Server passwords, CA keystore passwords, Database passwords) are derived from a *Cryptographic Secure Pseudo Random Number Generator* (CSPRNG) with at least 20 characters in length, taken from the following set [A-Za-z0-9] plus special characters with at least 3 digits, an example of such password could be: Uc3tg9oP~83#G42PeDo0\*CQL.

Weak passwords are a major issue when deploying a PKI, specially with RSA, as the research of (LHB<sup>+</sup>) on the sanity check of public keys collected on the web demonstrates.

### 3.1.5 Mutual Authentication

All authentication features used by the framework use mutual authentication with client certificates. These include access to the Web application private area, access to CA administration console, access to the Web application server administration console.

With client authentication, the web server authenticates the client by using the client's public key certificate.

Mutual authentication uses HTTP over SSL-TLS (HTTPS), in which the server authenticates the client using the client's public key certificate. SSL-TLS technology provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection. The certificate is issued by our own deployed CA, and provides identification for the bearer.

SSL-TLS secures communication by providing message encryption, integrity, and authentication. The SSL-TLS standard allows the involved components (such as browsers and HTTP servers) to negotiate which encryption, authentication, and integrity mechanisms to use, this *negotiation* can lead to issues (e.g. weak keys, insecure protocols, etc) as shown by the SSL-TLS installation assessment in 6.2.1.

- Encryption provides confidentiality by allowing only the intended recipient to read the message. SSL can use different encryption algorithms to encrypt messages. During the SSL handshake that occurs at the start of each SSL session, the client and the server negotiate which algorithm to use;
- Integrity ensures that a message sent by a client is received intact by the server, untampered. To ensure message integrity, the client hashes the message into a digest using a hash function and sends this message digest to the server. The server also hashes the message into a digest and compares the digests. Because SSL uses hash functions that make it computationally infeasible to produce the same digest from two different messages, the server can tell that if the digests do not match, then someone had tampered with the message;
- Authentication enables the server and client to check that the other party is who it claims to be. When a client initiates an SSL session, the server typically sends its certificate to the client. The client verifies that the server is authentic and not an impostor by validating the certificate chain in the server certificate. The server certificate is guaranteed by the certificate authority (CA) who signed the server certificate. The server also requires the client to have a certificate to authenticate the identity of the client.

When using certificate-based mutual authentication, the following actions occur:

1. A client requests access to a protected resource.
2. The web server presents its certificate to the client.
3. The client verifies the server's certificate.
4. If successful, the client sends its certificate to the server.
5. The server verifies the client's credentials.
6. If successful, the server grants access to the protected resource requested by the client.



Figure 3.6 illustrates the sequence of certificate-based mutual authentication.

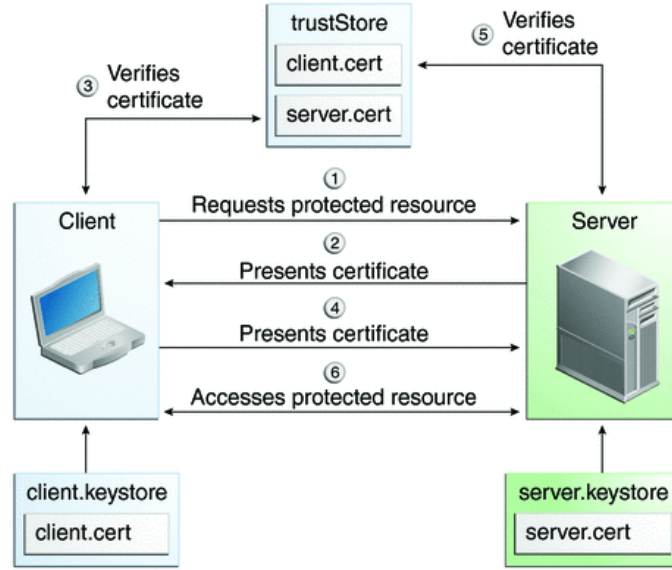


Figure 3.6: Client certificate-based mutual authentication sequence.

### 3.1.6 SCrypt Password Encryption

Along with mutual authentication and additional authentication step is required to access the private area of the framework, as modeled in the UML sequence diagram for login A.8. A password is created at the registration step, also modeled in the UML sequence diagram for registration A.8, this password is derived from the clients certificate keystore password.

We choose to have this additional authentication step since operating systems and browsers have their “unique” ways of dealing with client certificates. All operating systems and browsers ask for the client certificate upon import, but one can disable, enable or even force the certificate management software of several operating systems to ask for the certificate password whenever the certificate is used.

In our model we always wish to validate the user that is using the certificate so an additional layer of authentication is needed.

On registration the user needs to input their keystore certificate password, upon validation the inserted password is encrypted with *SCrypt* encryption algorithm and stored in the Web application database.

In our framework we also enforce some requirements to users chosen inputs for password, as shown by the research of (LHB<sup>+</sup>) on the sanity check of public keys collected on the web these issues shouldn’t be neglected.

An user input password needs at least to be 8 characters in length, with at least one upper case character, one digit and one special character.

The *SCrypt* encryption algorithm is a *Password-Based Key Derivation Function* (PBKDF) created by Colin Percival (Per09). The algorithm was specifically designed to make it costly to perform large scale custom hardware attacks by requiring large amounts of memory.

We will not formally illustrate the construction of the algorithm, this can be found at (Per09). A cost comparison of hardware to crack a password in 1 year, between several encryption and hash algorithms is also available in (Per09).

The *Scrypt* encryption algorithm takes 4 parameters:

- *Password* - is a parameter input in plain text;
- *N* - is the CPU cost parameter (default is  $2^{14} = 16384$  we use  $2^{16} = 65536$ );
- *r* - is the memory cost parameter (default is 8, we use 10);
- *p* - is the parallelization parameter (default is 1, we use 4).

## 3.2 Framework Requirements

These are the main requirements to the development of the framework:

- a CA that implements a PKI according to standards such as X.509 and IETF-PKIX and supports custom certificate extensions;
- a Web Application, accessible by Web browser, that enables end-users the “use” of *Secret Sharing* through their digital certificates, using the certificate extension as a *security-token*;
- the *security-token* supports integrity checks;
- access to the CA and the Web application must use “strong encryption” providing confidentiality and privacy;
- the framework should support implementation on a distributed orientated platform in a cloud and grid computing scenario.

The first requirement opens the discussion on which CA software should be considered.

A CA is the essential component of a PKI and its primary function is the management of digital certificates, to issue and revoke certificates. These may be issued for different goals like: validate users and devices, secure communication with *Secure Socket Layer* (SSL)-*Transport Layer Security* (TLS) for clients and servers, signature and email encryption, signing documents, access to systems by using cryptographic cards, secure VPN connections and many more.

Two choices emerged from a preliminary analysis, EJBCA (ejba) and OpenCA(ope), table 3.1 shows a comparison between EJBCA and OpenCA considering different parameters (Vat11). OpenCA is based on many Open-Source Projects, among these there are OpenLDAP, OpenSSL and Apache Project.

*Enterprise Java Beans Certificate Authority* (EJBCA) is supported by a Swedish company PrimeKey Solutions AB<sup>®</sup>, which holds the copyright to most of the codebase.

CA-PKI Software		
	<i>OpenCA</i>	<i>EJBCA</i>
<b>Supported O.S.</b>	Linux Flavors	Platform Independent
<b>Implementation</b>	C, Javascript, Perl, Unix Shell	Java
<b>Modules</b>	Perl	<i>Enterprise JavaBeans</i> (EJB)
<b>LDAP</b> <sup>3</sup>	Yes	Yes
<b>CRL</b> <sup>4</sup>	Yes	Yes
<b>OCSP</b> <sup>5</sup>	Yes	Yes
<b>Algorithm</b>	Can be chosen	Can be chosen
<b>Scalability Support</b>	Not really	Yes
<b>Configuration</b>	Complex	Even more complex
<b>License</b>	BSD-3	LGPL
<b>Cost</b>	Free	Free

Table 3.1: Comparison between CA-PKI software

Analyzing the previous table 3.1 we can verify that OpenCA software doesn't support scalability, being that one of the requirements states that the framework should support a cloud and grid computing scenario, the choice of CA software falls on EJBCA.

We will not go in further technical detail why this was the choice of CA-PKI, but simply state that its used by inumerous high profile organizations such as: National Swedish Police Board; France Ministry of Defense; Multicert e-Citizen card, Portugal; and many more. EJBCA supports the technologies needed to implement our framework, such as X.509 certificate extensions and can be deployed in a clustered, scalable environment.

## 3.3 PKI CA Software

### 3.3.1 EJBCA Features

These are some of the features presented by EJBCA. We will not enumerate all of the features presented by EJBCA but only state the ones that are most relevant to this work.

#### EJBCA PKI System Features

- Multiple CA's and levels of CA's, build a complete infrastructure (or several) within one instance of EJBCA;
- Follows X.509 and PKIX (RFC-5280);
- Issue SSL/TLS certificates that work with most common servers;

<sup>3</sup>Lightweight Directory Access Protocol (LDAP)

<sup>4</sup>Certificate Revocation List (CRL)

<sup>5</sup>Online Certificate Status Protocol (OCSP)

- Server and Client certificates can be exported as *Public Key Cryptographic Standard* (PKCS)#12, *Java KeyStore* (JKS) or *Privacy-enhanced Electronic Mail* (PEM);
- Browser enrollment;
- Revocation and CRL's;
- CRL creation and URL-based CRL distribution points according to RFC-5280;
- Configurable certificate profiles for different types and contents of certificates;
- Standard and custom certificate extensions supported;
- Supports *Simple Certificate Enrollment Protocol* (SCEP);
- Supports OCSF including *Authority Information Access* (AIA) extensions (RFC-2560 and RFC-5019);
- Supports RFC-4387 for distribution of CA certificates and CRL's over HTTP;
- Validation Authority service serving OCSF responses (RFC-2560/5019), CA certificates and CRL's (RFC-4387);
- Supports *Certificate Management Protocol* (CMP) (RFC4210 and RFC4211);

### **EJBCA Integration Features**

- External *Validation Authority* (VA) and OCSF responder can run integrated with EJBCA or stand-alone (clustered), for high-performance and high-availability;
- Web service interface for remote administration and integration;
- *Application Programming Interface* (API) for an external RA, restricting in-bound traffic to CA;

### **EJBCA Administration Features**

- Configurable entity profiles for different types of users;
- Random or manual password for initial user authentication;
- Multiple levels of administrators with specified privileges and roles;
- Store Certificates and CRL's in SQL database, LDAP and/or other custom data source;
- Component based architecture for various authorization methods of entities when issuing certificates;
- Simple stand-alone batch enrollment *Graphical User Interface* (GUI) for *Certificate Signing Request* (CSR)'s (Web Service RA);

### 3.3.2 EJBCA Requirements

The requirements needed to build and run EJBCA are:

- *Java Development Kit* (JDK) 1.6 OpenJDK or Oracle® JDK, OpenJDK is recommended;
- *Java Cryptography Extension* (JCE) Unlimited Strength Jurisdiction Policy Files (for Oracle's® JDK , not required for OpenJDK); <sup>6</sup>
- Java Application Server, JBoss 5.1.0 is the recommended Server; <sup>7</sup>
- Apache Ant 1.7.1 or later. <sup>8</sup>

## 3.4 Conclusions

Stated in the features of the CA PKI chosen for development of our framework we denote EJBCA follows RFC-5280 which is part of the X.509v3 specification, and states in section 4.2 the following: “The extensions defined for X.509 v3 certificates provide methods for associating additional attributes with users or public keys and for managing relationships between CAs. The X.509 v3 certificate format also allows communities to define private extensions to carry information unique to those communities.”

The previous statements consolidates the premise that it is plausible to define/build a custom certificate extension to carry unique information.

---

<sup>6</sup>EJBCA makes use of strong cryptography and Keystore passwords must be longer than 7 characters. For this to work you must install the *Unlimited Strength Jurisdiction Policy Files* for JDK.

<sup>7</sup>Other Java Application Servers are currently supported as Glassfish 2.1.1 and WebLogic 10.3.4, but your mileage may vary.

<sup>8</sup>Javascript support may be needed in Ant for some components.



# Chapter 4

## Architecture

### Contents

---

<b>3.1 Preliminaries . . . . .</b>	<b>15</b>
3.1.1 Object Identifier (OID) . . . . .	15
3.1.2 Abstract Syntax Notation One (ASN.1) . . . . .	15
3.1.3 One-Way Functions . . . . .	17
3.1.4 Pseudo Random Number Generators . . . . .	18
3.1.5 Mutual Authentication . . . . .	20
3.1.6 SCrypt Password Encryption . . . . .	21
<b>3.2 Framework Requirements . . . . .</b>	<b>22</b>
<b>3.3 PKI CA Software . . . . .</b>	<b>23</b>
3.3.1 EJBCA Features . . . . .	23
3.3.2 EJBCA Requirements . . . . .	25
<b>3.4 Conclusions . . . . .</b>	<b>25</b>

---

In this chapter we show the architectural model of our framework. As well as the developed distribution/recover algorithms that support our “proof of concept” framework.

## 4.1 Communication Model between CA and Web Application

As stated previously EJBCA supports a Web service interface for remote administration and integration. It also supplies a fully working RA sample program, written in *Java Standard Edition* (JSE) that uses EJBCA Web Service (ejbb) for communication with the EJBCA applicational server. It includes most of the operations needed to manage certificates, depending of the privileges of the RA.

This program will serve as the basis for the development of an JSE application which will provide the following operations:<sup>1</sup>

- user edit;
- distribution of secrets;
- recovery of secrets;

All of the previous operations implicitly issue new certificates to end-users. So, all values updated by the job/task done by the Web service will be reflected in the new issued certificate.

Our communication model approach uses the database, with a table in which certain fields are used for synchronization between the Web application and the Web service RA application.

As shown in the *Entity Relationship Diagram* (ERD) A.13 of the database there’s a table named *s\_job* this is the table used to synchronize events between both applications.

---

<sup>1</sup>All operations implicitly issue new certificates to end-users.

Following are the main fields used by both applications and their purpose.

- Field *group\_name* is to find users belonging to the a certain group (used in distribution/recovery jobs);
- Field *user* is to find a corresponding user (only used in edit jobs);
- Field *job\_status* is for error handling, web service status and job status, it uses 8 enumerated values;
  1. QUEUE;
  2. BUSY;
  3. DONE;
  4. EJBCA\_WS\_DOWN;
  5. EJBCA\_WS\_UP;
  6. ERROR\_USER\_EDIT;
  7. ERROR\_DIST;
  8. ERROR\_REC;
- Field *job\_description* is used to define what type of job to execute, it has 3 enumerated values;
  1. EDIT;
  2. DISTRIBUTE;
  3. RECOVER;
- Field *job\_data* contains the data, in binary form, to be encrypted (only used in distribution jobs);

The Web service RA application periodically checks the table *s\_job* for jobs/tasks, this is accomplished by a *ScheduledExecutorService*.

A *ScheduledExecutorService* is a Java interface class that extends *ExecutorService*. “An *ExecutorService* can schedule commands to run after a given delay, or to execute periodically” (java).

We instantiate the *ScheduledExecutorService* with the factory method *newScheduledThreadPool* (javb), this method takes as argument an integer that defines the number of threads in the thread-pool, this value should be parameterized according to the host machine architecture (e.g. in an 8 core machine, 6 threads framework jobs, 2 threads for garbage collecting).

In our Web service RA application we instantiate two *ScheduledExecutorServices*, one for edit, distribute and recovery jobs and another for garbage collection of jobs so that the database table doesn't fill up with already finished jobs/tasks.

Both *ScheduledExecutorService* run/execute with a fixed delay. The *ScheduledExecutorService* for edit, distribute, recovery jobs/tasks runs every 960ms and the *ScheduledExecutorService* for garbage collection runs every 144 960ms (2.4 min. approximately).



The following figure 4.1 shows an schematic of the communication model between EJBCA-CA and the Web application using the EJBCA-RA Web service.

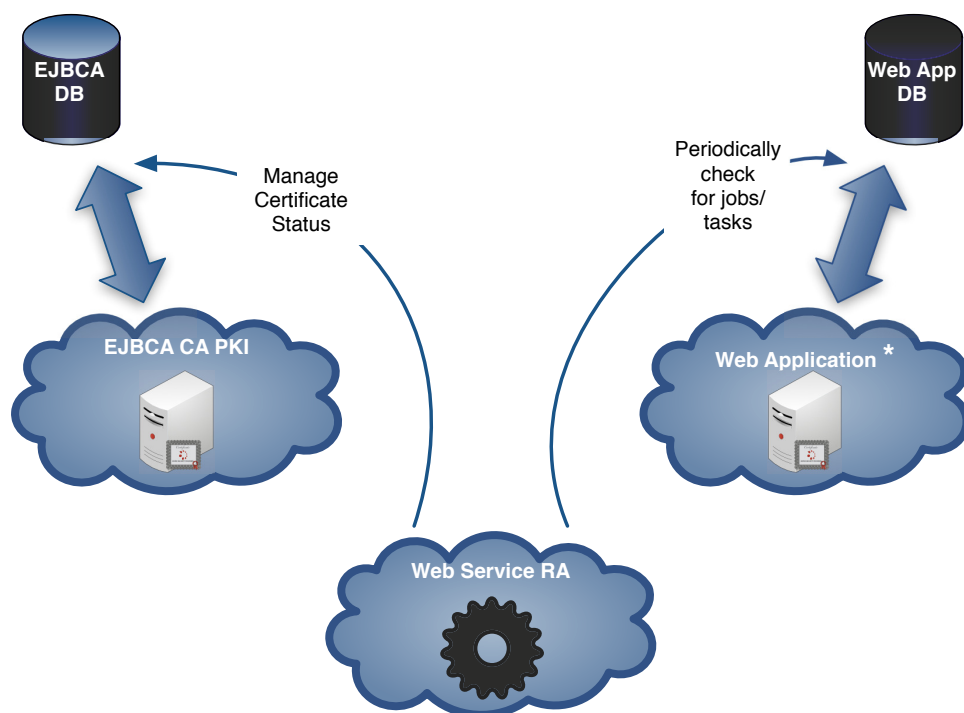


Figure 4.1: Communication model between EJBCA-CA and the Web application using the Web service.

The RA Web service is the entity in charge of the management of *secret distribution*, *secret recovery* and user certificates. This application is the *trusted third party* in our model.

## 4.2 Secret Distribution

In this section we'll show the procedures that lead to the secret distribution, as well as the algorithm implemented for secret sharing.

As Shamir's SSS works on finite fields, more specifically  $\mathbb{Z}_p$ , we need to clarify the boundaries on this field, as all other values used by the distribution algorithm are related to these boundaries, like the secret  $\mathcal{S}$  that's going to be distributed and the cardinality of participants  $|n|$ .

The most important specification is the size of *prime*, to form  $\mathbb{Z}_p$ , to use modular arithmetic. In our work we use a 160 bit  $p$  and a 156 bit  $\mathcal{S}$ .

The process that leads to the creation of a *secret distribution* job/task in the DB, that will be posteriorly caught by the RA Web service ( $\mathcal{TTP}$ ) is illustrated in the following figure 4.2.

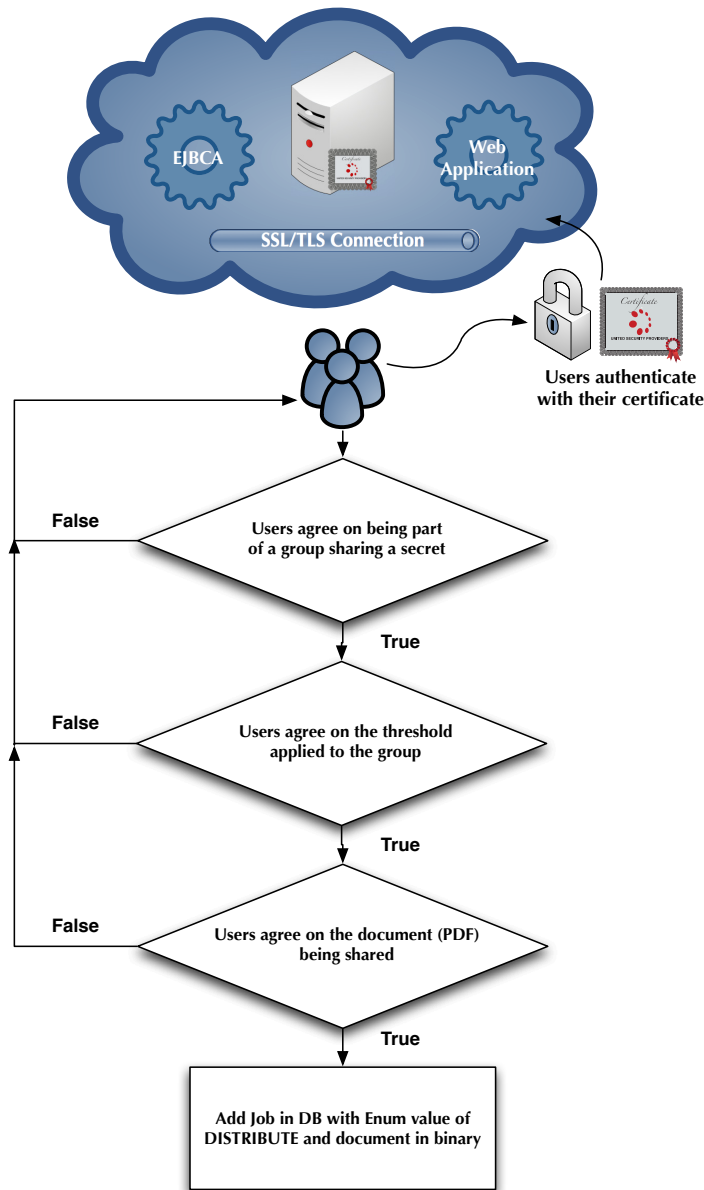


Figure 4.2: Process that leads to the creation of a secret distribution job.

When the  $\mathcal{TTP}$  “picks up” a new secret distribution job for a group the following algorithm 1 is processed.

---

**Algorithm 1** Secret sharing algorithm

---

**Require:** instantiate a CSPRNG, to choose  $p$ , polynomial coefficients and  $\mathcal{S}$

- 1:  $\mathcal{TTP}$  chooses  $\mathcal{S}$  from  $\mathbb{Z}$
  - 2:  $\mathcal{TTP}$  chooses prime  $p > n \wedge p > S$  ( $n$  cardinality of users)
  - 3:  $\mathcal{TTP}$  selects  $k - 1$  random, independent coefficient  $a_1, \dots, a_{k-1}$ , such that  $0 \leq a_j \leq (p - 1)$ , defining a random polynomial over  $\mathbb{Z}_p$ ,  $f(x) = \sum_{j=0}^{k-1} a_j x^j$
  - 4: **for all**  $1 \leq i \leq n$  (users that belong to a group of cardinality  $n$ ) **do**
  - 5:    $\mathcal{TTP}$  computes  $S_i = f(i) \bmod p$
  - 6:    $\mathcal{TTP}$  gets *salt* from CSPRNG and compute *salted*  $S_i = \textit{salt} + S_i$
  - 7:    $\mathcal{TTP}$  hash SHA-256(*salted*  $S_i$ )
  - 8:    $\mathcal{TTP}$  converts *salted*  $S_i$  to Hexadecimal
  - 9:    $\mathcal{TTP}$  updates Web application DB with *user index*, hashed *salted*  $S_i$  (Hex), *salt* (Hex)
  - 10:   **if** user certificate extension value is **null** **then**
  - 11:      $\mathcal{TTP}$  add values group identifier,  $S_i$  (plaintext) and hashed *salted*  $S_i$  (Hex) to user *certificate extension*
  - 12:   **end if**
  - 13:   **if** user certificate extension value **not null** **then**
  - 14:      $\mathcal{TTP}$  concatenates values group identifier,  $S_i$  (plaintext) and hashed *salted*  $S_i$  (Hex) to user *certificate extension*
  - 15:   **end if**
  - 16:    $\mathcal{TTP}$  sets user client certificate status to *New*, in EJBCA DB, for posterior certificate request
  - 17: **end for**
  - 18:  $\mathcal{TTP}$  gets *salt* from CSPRNG and computes *salted*  $\mathcal{S} = \textit{salt} + \mathcal{S}$
  - 19:  $\mathcal{TTP}$  hashes *salt* and *salted*  $\mathcal{S}$ , SHA-256(*salt*), SHA-256( $\mathcal{S}$ )
  - 20:  $\mathcal{TTP}$  converts *salt* and *salted*  $\mathcal{S}$  to Hexadecimal
  - 21:  $\mathcal{TTP}$  gets group PDF document in binary, from Web application DB, and encrypts PDF with AES-256 using  $\mathcal{S}$
  - 22:  $\mathcal{TTP}$  updates Web application DB with group *modular arithmetic*, *salt* (Hex), *salted*  $\mathcal{S}$  (Hex), *encrypted PDF* (in binary)
-

### 4.3 Secret Recovery

The process that leads to the creation of a *secret recovery* job/task in the DB, that will be posteriorly caught by the RA Web service ( $\mathcal{TTP}$ ) is illustrated in the following figure 4.3.

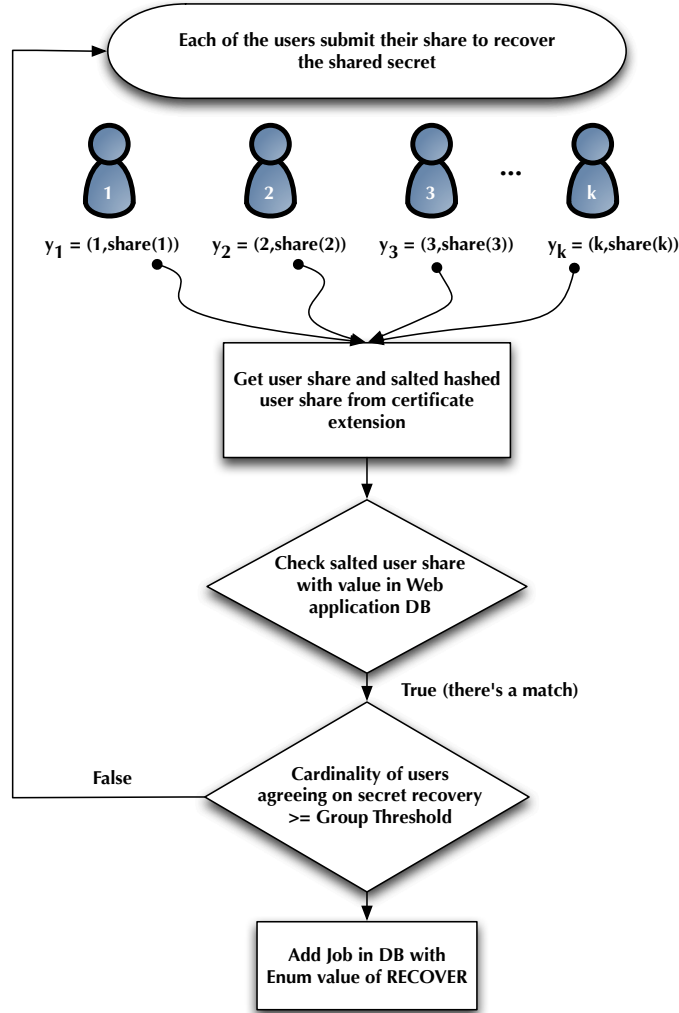


Figure 4.3: Process that leads to the creation of a secret recovery job.

When the  $\mathcal{TTP}$  “picks up” a new secret recovery job for a group the following algorithm 2 is processed.

---

**Algorithm 2** Secret recovery algorithm

---

```
1:  $\mathcal{TTP}$  get group identifier from Web application DB
2:  $\mathcal{TTP}$  get list of users that belong to previous group and want to recover the group secret ( $n$ )
3: if cardinality of users that wish to recover the secret  $n \geq k$  (group threshold from Web
   application DB) then
4:    $\mathcal{TTP}$  gets modular arithmetic value from Web application DB
5:   for all  $1 \leq i \leq n$  (users that belong to a group of cardinality  $n$ ) do
6:      $\mathcal{TTP}$  process Lagrange interpolation
7:      $\mathcal{TTP}$  get user index from Web application DB
8:      $\mathcal{TTP}$  get user share from Web application DB
9:   end for
10:   $\mathcal{TTP}$  gets recovered secret (BigInteger)
11:   $\mathcal{TTP}$  gets salt (Hex) for group secret from Web application DB
12:   $\mathcal{TTP}$  gets salted  $\mathcal{S}$  (Hex) for group secret from Web application DB
13:   $\mathcal{TTP}$  converts salt from Hex to byte array
14:   $\mathcal{TTP}$  converts recovered secret from Hex to byte array
15:   $\mathcal{TTP}$  joins arrays salted recovered secret = salt + recovered secret
16:   $\mathcal{TTP}$  processes SHA-256(salted recovered secret)
17:   $\mathcal{TTP}$  gets group secret hash (in Hex) and converts to byte array
18:  if integrity check OK (digests of salted recovered secret = group secret hash) then
19:     $\mathcal{TTP}$  gets encrypted group secret (PDF) (in binary)
20:     $\mathcal{TTP}$  decrypt encrypted group secret(PDF) with recovered secret
21:     $\mathcal{TTP}$  set decrypted PDF (in binary) in Web application (value will be cleared when last
       user that belongs to group logs-off)
22:  end if
23:  if integrity check FAILS (digests of salted recovered secret  $\neq$  group secret hash) then
24:     $\mathcal{TTP}$  marks job/task with error
25:     $\mathcal{TTP}$  clears unnecessary values
26:  end if
27: end if
28: if cardinality of users that wish to recover the secret  $n < k$  (group threshold from Web
   application DB) then
29:    $\mathcal{TTP}$  marks job/task with error
30:    $\mathcal{TTP}$  clears unnecessary values
31: end if
```

---



# Chapter 5

## Implementation

### Contents

4.1	Communication Model between CA and Web Application . . . . .	27
4.2	Secret Distribution . . . . .	30
4.3	Secret Recovery . . . . .	32

In this chapter we go through the steps of EJBCA installation, build of our own CA hierarchy, as well as define procedures for securing the applicational servers JBoss and Glassfish, as well as the RDBMS MySQL.

We also denote some Web application technologies considerations.

We will discuss some *use cases* for the considered cryptographic approaches, PKI with SSS, and enunciate our own use case.

## 5.1 EJBCA Deployment Process

### 5.1.1 Securing MySQL RDBMS

The installation of the RDBMS MySQL is pretty straight forward on major operating systems so we will not describe it here.

In order to achieve the highest possible level of security, the installation and configuration of MySQL should be performed in accordance with the following security requirements (mys):

- MySQL databases must be executed in a *chrooted* environment <sup>1</sup>;
- MySQL processes must run under a unique UID/GID that is not used by any other system process <sup>2</sup>;
- MySQL root's account must be protected by a hard to guess password;
- Only local access to MySQL will be allowed <sup>3</sup>;
- The administrator's account will be renamed <sup>4</sup>;
- Anonymous access to the database (by using the nobody account) must be disabled;
- All sample databases and tables must be removed.

Some of these steps to ensure MySQL security can be achieved by running the script `mysql_secure_installation`, such steps include: removing test/sample databases, removing

---

<sup>1</sup>this wasn't performed.

<sup>2</sup>this wasn't performed, but some O.S. already assure this.

<sup>3</sup>We limited MySQL to use only local network access by adding the following to MySQL configuration file: `bind-address = 127.0.0.1` and created a rule in the firewall blocking all non-local traffic.

<sup>4</sup>this wasn't performed

anonymous accounts and securing the root user with a chosen password <sup>5</sup>.

### 5.1.2 Securing JBoss Server

These are the recommended steps in securing JBoss when deploying EJBCA (ejba).

- Remove `Root.war` from deploy folder;
- Remove all files from management folder, in the deploy folder;
- Remove administration console application by removing the `admin-console.war` file in the deploy folder;
- Remove JMX console application by removing the `jmx-console.war` file in the deploy folder;
- Remove the HTTP invoker by removing the `http-invoker.sar` file in the deploy folder;

Since EJBCA will be the only enterprise application deployed to JBoss removing these management features won't be an issue and will improve overall security.

### 5.1.3 EJBCA Installation

EJBCA installation configuration is done by editing properties files in the `conf` directory. The configuration file samples provide plenty of comments describing various options.

Transcripts of all the property files used in our configuration of EJBCA can be found in A.1.

We will deploy EJBCA “vanilla” flavored as we will later discard the the initial CA and build our own CA hierarchy.

We only did minor changes to the initial “vanilla” configuration files, these changes include:

- enabling the certificate store servlet (default is disabled), in the `certstore.properties` file;
- enabling the CRL store servlet (default is disabled), in the `crlstore.properties` file;
- change/configure the RDBMS to MySQL<sup>®</sup>, in the `database.properties` file;
- change CA server DNS hostname, according to our DNS, limit the log maximum query row count to 100 (default is 1000), change the support languages of the Web GUI EJBCA to only English and Portuguese (by default 9 languages are supported), in the `web.properties` file;
- change the OCSP response servlet signature algorithm to `SHA256WithRSA` (defaults are `SHA1WithRSA,SHA1WithECDSA;SHA1WithDSA`), in the `ocsp.properties` file;
- change the amount of free memory before alarming to 32 Mb (default is 1 Mb) to EJBCA Health-check servlet, enable CA tokens to actually perform a signature test (default is false), change the *BCrypt* password log rounds to 16 <sup>6</sup> (default is 1), in the `ejbca.properties` file;
- change the signature algorithm of the CA to `SHA256WithRSA` (default is `SHA1WithRSA`), change the validity of the initial CA to 1 day <sup>7</sup> (default is 3650 days, 10 years), in the

---

<sup>5</sup>this script is not available on Windows O.S.

<sup>6</sup>To better protect from off-line brute force attacks of passwords on a compromised database, the computationally expensive *BCrypt* algorithm can be configured to use higher log-rounds values that will increase computational cost. Values can be between 1,...,31. “A decent value for high security is 8”. According to EJBCA sample file.

<sup>7</sup>As this certificate is only being used to deploy EJBCA for the first time, so it only has 1 day validity. We will build our CA hierarchy with other certificates.



`install.properties` file.

## Custom Certificate Extension

One important file in our configuration is the property file that specifies the *certificate extensions* to be used by EJBCA when issuing certificates.

As stated previously our objective is to provide a CA that embeds a *security token* in the issued digital certificates to support *Threshold Cryptography*, in this case Shamir's SSS. The previous *custom certificate extension* enables this.

The certificate extension has the following requirements:

- an unique specified OID of the extension<sup>8</sup>;
- full class name (classpath) of the certificate extension implementation class<sup>9</sup>;
- specification if the certificate extension should be used or be disabled;
- specification if the display name should be translated in the language resources;
- specification if the extension should be marked as critical in the certificate;
- specification if the extension encoding value and if this value is set dynamically or not<sup>10</sup>.

This certificate extension will hold, specifically, 3 values:

- a group identifier;
- user share in plain-text, for this group;
- a *salt hashed* value of the user share, for this group.

All of these values, also use string terminator symbols, for posterior parsing of the DERUTF8String. Group identifier uses a terminator string symbol of ':' and the user shares use the terminator string symbol of ';'.

As our certificate extension can hold several user shares, for several groups, we need a way to know where a user share ends and another begins.

The following figure 5.1 illustrates an example of such certificate extension.

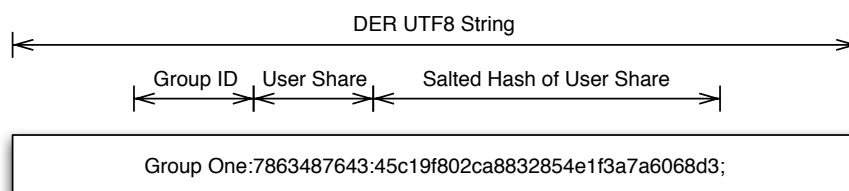


Figure 5.1: Certificate extension illustration.

<sup>8</sup>We use OID 2.999, it can be used by anyone, without any permission, for the purpose of documenting examples of OIDs .

<sup>9</sup>In the case of EJBCA basic certificate extension this is:  
`org.ejbca.core.model.ca.certextensions.BasicCertificateExtension`.

<sup>10</sup>In our case we use DERUTF8String.

## EJBCA Installation Procedure

Before installing we should make sure the right java tools (`javac/keytool`) are available in your system `PATH` (i.e. `/usr/local/jdk1.6.0_xx/bin`) <sup>11</sup>.

Due to Java's memory handling you may need to assign more memory to `Ant` in order to build the system without `OutOfMemory` errors. You can do that by setting an environment variable: `ANT_OPTS=-Xmx512m`.

There are also a few *bugs* that need workaround in order to correctly install EJBCA.

- bug with *Bouncy Castle* libraries in EJBCA;  
**workaround** - copy `EJBCA_HOME/lib/bc*.jar` to `JBOSS_HOME/server/default/lib/`
- on JBoss 5.1.x the *Web Service Definition Language* (WSDL) location gets incorrectly generated by default;  
**workaround** - edit the file located at:  
`APPSRV_HOME/server/default/deployers/jboss/ws.deployer/META-INF/jboss-beans.xml`  
and comment the line `<property name="webServiceHost">$jboss.bind.address</property>`

These are the required steps to install EJBCA.

1. open a **console (terminal)** in the EJBCA directory/folder and type: `ant bootstrap`.  
This will compile and build EJBCA and deploy it to JBoss;
2. open a **console (terminal)** and start JBoss, with the command `run.sh`;
3. open a **console (terminal)** in the EJBCA directory/folder and type: `ant install`.  
This will generate all certificates, keys, etc, needed to run with the initial CA. Administrator client certificate is found in `EJBCA_HOME/p12` this will be imported to the browser for EJBCA administration;
4. stop JBoss. If JBoss is running from **console (terminal)** just press `Crt+C`;
5. open a **console (terminal)** in the EJBCA directory/folder and type: `ant deploy`. This will deploy everything again and configure the servlet container with the keystore file (this is why we needed to stop the container);
6. import the certificate from `EJBCA_HOME/p12/superadmin.p12` in your web browser. This is the super administrator certificate used to access the administrator GUI;
7. start JBoss again, after this step EJBCA installation is finished. One can now access the administrator GUI at `https://EJBCA_hostname:8443/ejbca/` , or the public web pages at `http://EJBCA_hostname:8080/ejbca`.

### 5.1.4 CA Hierarchy Implementation

After installing EJBCA we will build our own CA hierarchy. Discard the original root CA and issue new server certificates for EJBCA and client certificates for EJBCA administration. The following diagram 5.2 illustrates the hierarchy we will build.

---

<sup>11</sup>Only JDK 1.6 versions are supported, JDK 7 isn't supported.

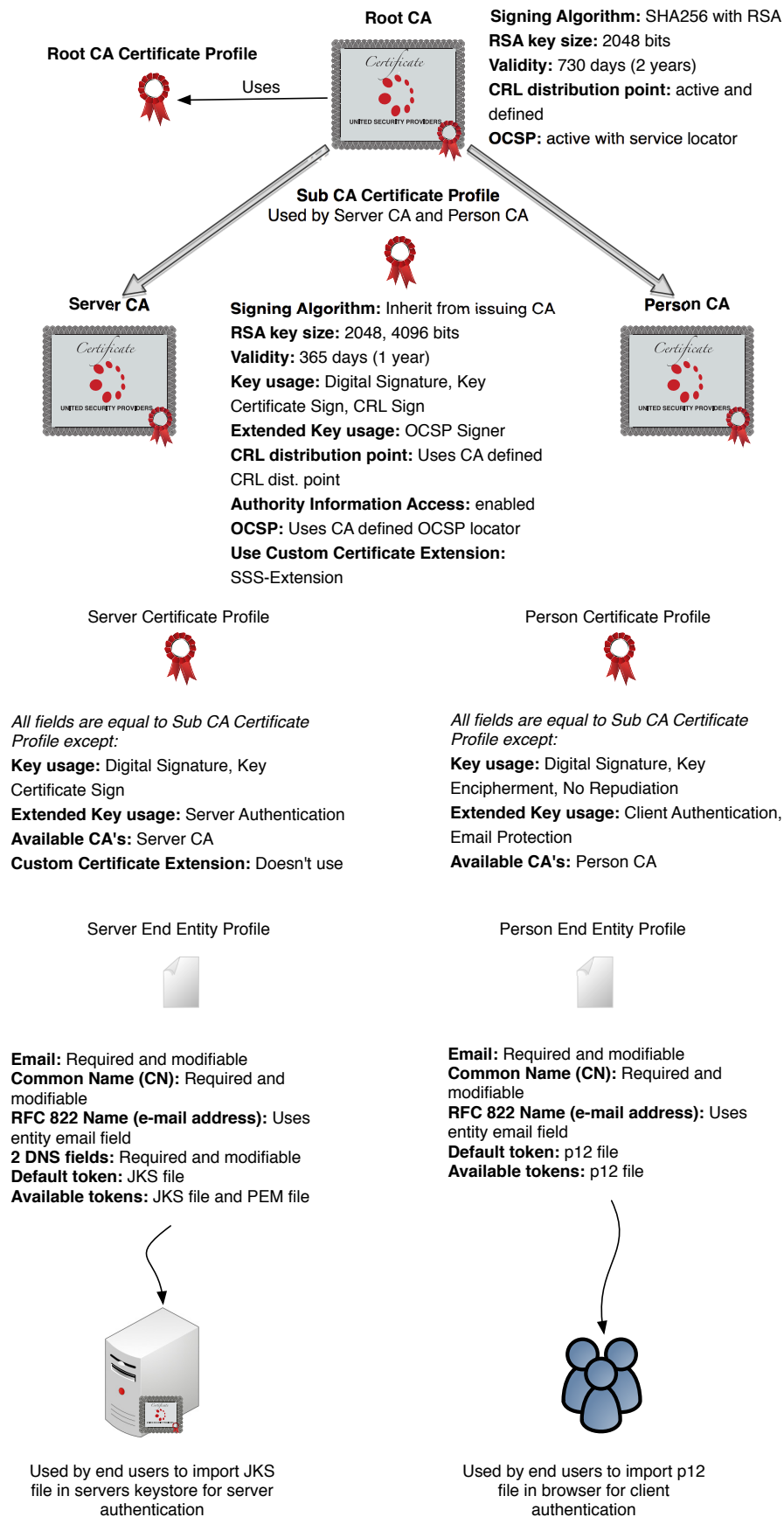


Figure 5.2: EJBCA hierarchy diagram

These are the steps required to setup the new *hierarchy*:

1. Issue new super-administrator key and certificate. Create new *Super Administrators Group*;
  - (a) In EJBCA administrator Web GUI go to *Administration* → *Add End Entity Page* select *Person End Entity Profile*. Fill in the required information;
  - (b) Go to EJBCA *Public Web* → *Create Browser Certificate* and provide the credentials previously specified for the administrator end entity;
  - (c) Import new super-administrator certificate into the browser;
  - (d) In EJBCA administrator Web GUI go to *Administration* → *Edit Administrator Privileges* click *Add*, give it a name (Super Administrators)
  - (e) Provide the required information to specify the new administrator certificate;
2. Prepare new Keystore and Truststore for EJBCA server;
  - (a) In EJBCA administrator Web GUI go to *Administration* → *Add End Entity Page* select *Server End Entity Profile*. Fill in the required information. The password should must match the password specified in the `web.properties` file A.1.7 for the `httpsserver.password` parameter;
  - (b) Generate the new Keystore, on a console (terminal), with the following commands:  
`ejbca.sh batch`
  - (c) Generate the new Truststore, on a console (terminal), with the following commands:  
`ejbca.sh ca getrootcert 'Root CA' rootca.der -der`  
`ejbca.sh ca getrootcert 'Person CA' personca.pem`  
`openssl x509 -in personca.pem -out personca.der -outform DER`  
`keytool -importcert -alias rootca -file rootca.der -keystore ↵`  
`truststore_new.jks`  
`keytool -importcert -alias personca -file personca.der -keystore ↵`  
`truststore_new.jks`
3. Configure CRL updater service. CRL updater service is in charge of generating the CRL's and making sure that once they expire they get regenerated. It is executed periodically;
  - (a) In EJBCA administrator Web GUI go to *Administration* → *Edit Services*
  - (b) Select CA's to check. Select periodical interval. Activate service.

Before proceeding with deployment of new Keystore and Truststore, make sure to stop the EJBCA. Deploy the new Keystore and Truststore by copying (or replacing) the files in JBoss directory/folder *keystore*.

## 5.2 Web Application Technologies

In regard to the Web technology for the development of the Web application, we've chosen JEE technology, but as stated earlier and considering the communication model between the Web application and the CA this choice isn't unique and one could consider other Web application development technologies, such as: PHP, .NET, Ruby, etc.

For the development of our Web application we used the following technologies:

- GlassFish Server Open Source Edition 3.1.2
- *Java Server Faces* (JSF) (a Java specification for building component-based user interfaces for web applications)
- Primefaces (Ajax framework with JSF components)
- Omnifaces (utility library for JSF that focusses on utilities that ease tasks with JSF API)
- Atmosphere Framework (client and server side components for building Asynchronous Web Application)
- iText is an open source library that allows you to create and manipulate PDF documents <sup>12</sup>

We will not specify all of procedures to develop the Web application as this isn't the focus of this thesis, but we will mention more relevant procedures regarding *security* and *certificate extension* use.

### 5.2.1 Securing GlassFish Server

In order to secure Glassfish server we followed the following procedures:

- change the *master password*. Glassfish uses it to protect the domain-encrypted files from unauthorized access, i.e. the certificate store which contains the certificates for https communication;
- change *administration password*, this is password bound to a user (admin), that starts a domain or has privileges to access the administrator console;
- issue a new server certificate in EJBCA, this step is similar to 2a in CA Hierarchy Implementation section, but the password must match the previous *master password*;
- issue a client certificate for server administration in EJBCA, this step is similar to 1b in CA Hierarchy Implementation section, but the password must match the previous *master password*;
- import client certificate certificate chain to previously issued server certificate;
- import client certificate to the browser;
- go to Glassfish administrator console, and activate *secure administration*. This enforces administration console to use mutual authentication with the previously deployed certificates.

---

<sup>12</sup>Also used by Web service RA

### 5.2.2 Framework on the Web

Our framework is currently located at `secret-sharing.tk` or `www.secret-sharing.tk`.

### 5.2.3 Web Application Workflow

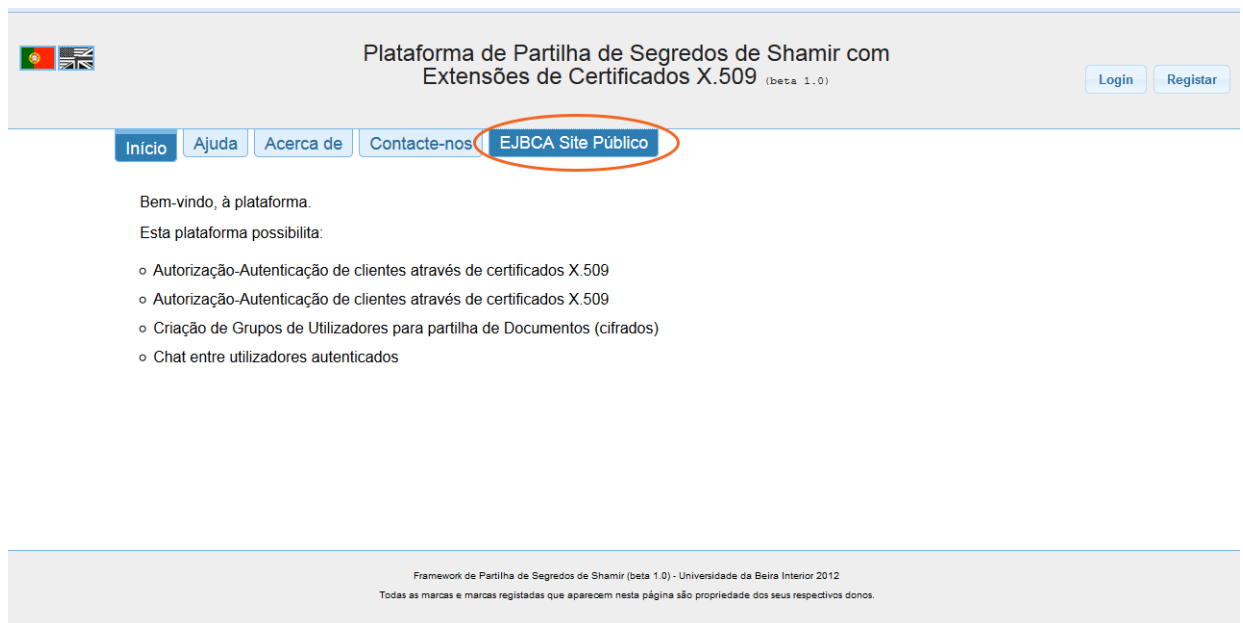


Figure 5.3: Web application main window

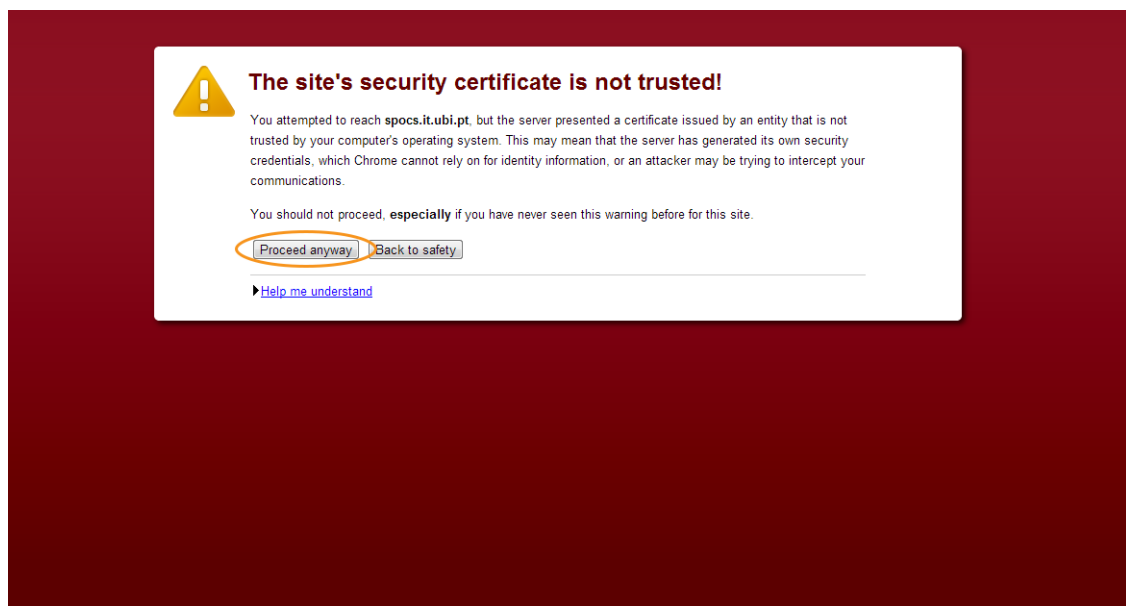


Figure 5.4: Firefox browser warning

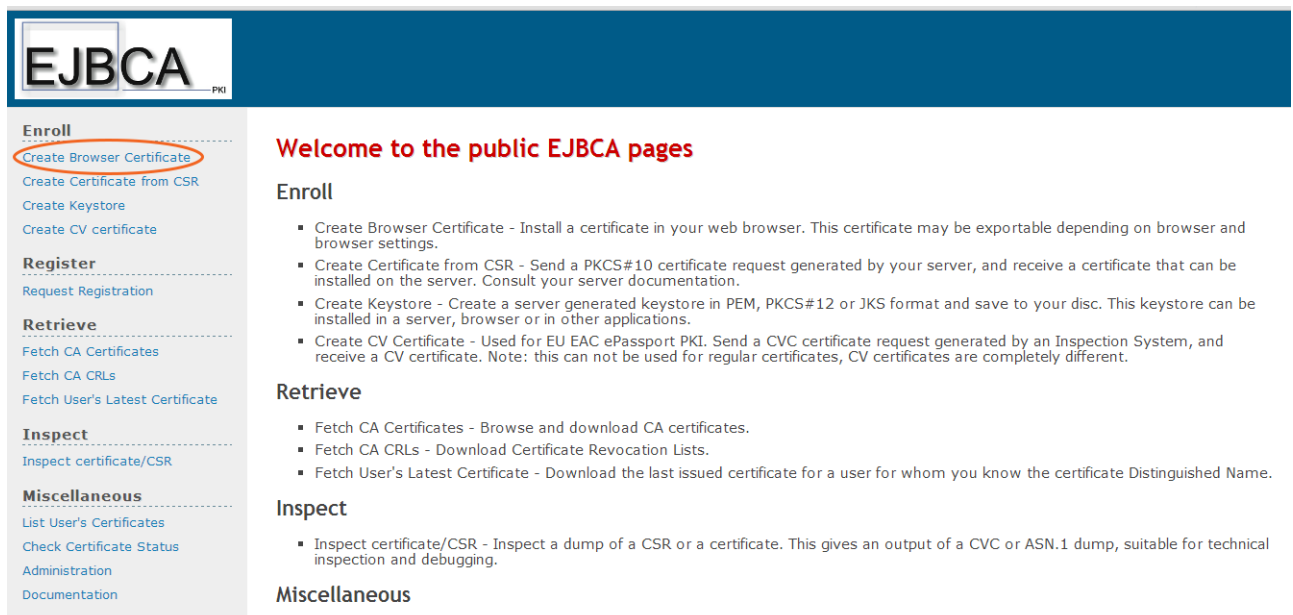


Figure 5.5: EJBCA public site

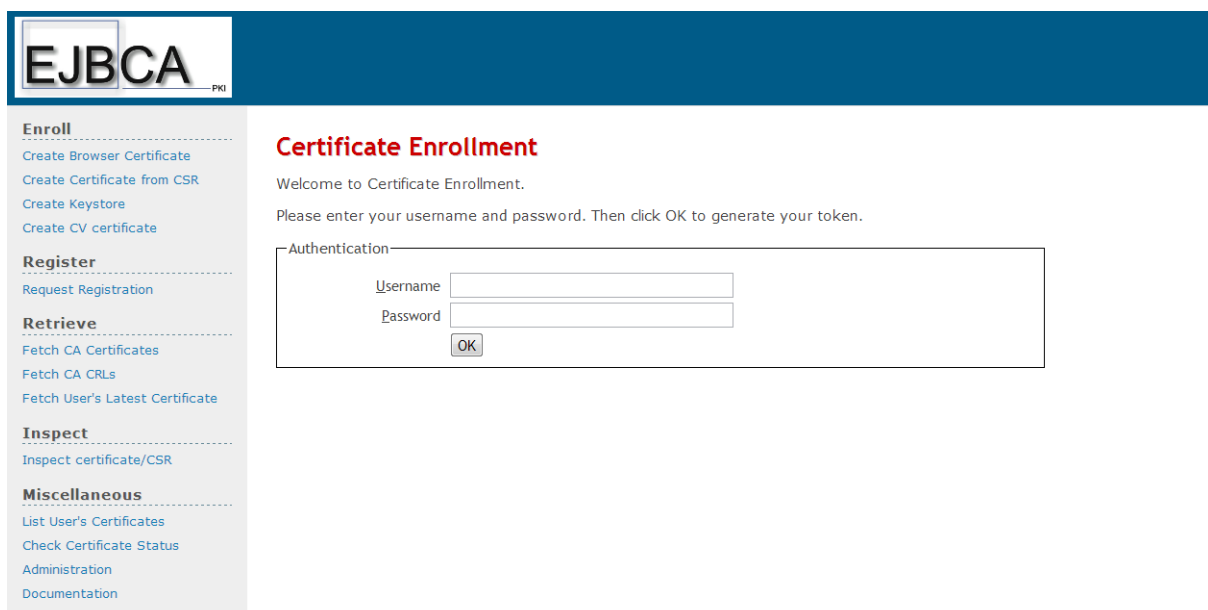


Figure 5.6: EJBCA certificate enrollment/request form





Figure 5.7: Web application login request

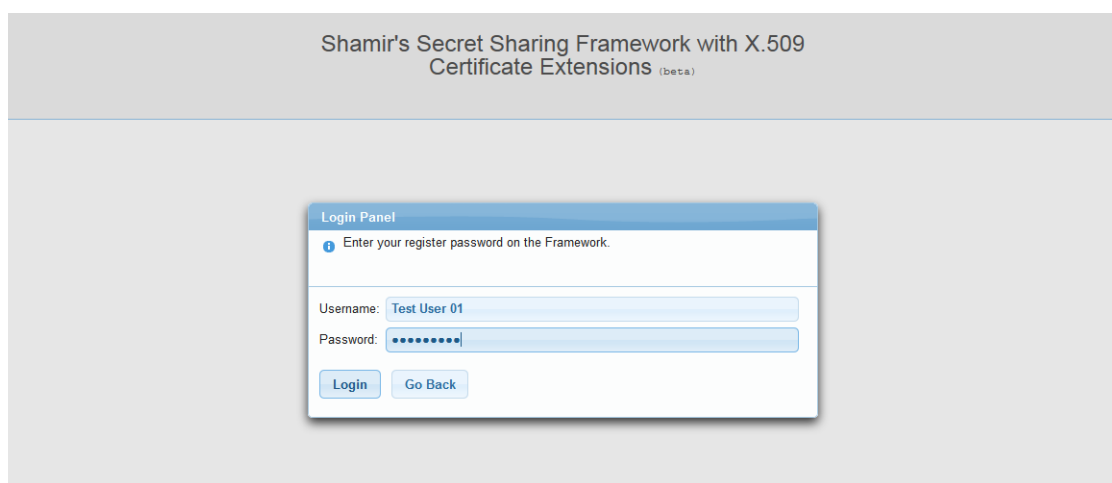


Figure 5.8: Web application login form



Figure 5.9: Web application login success

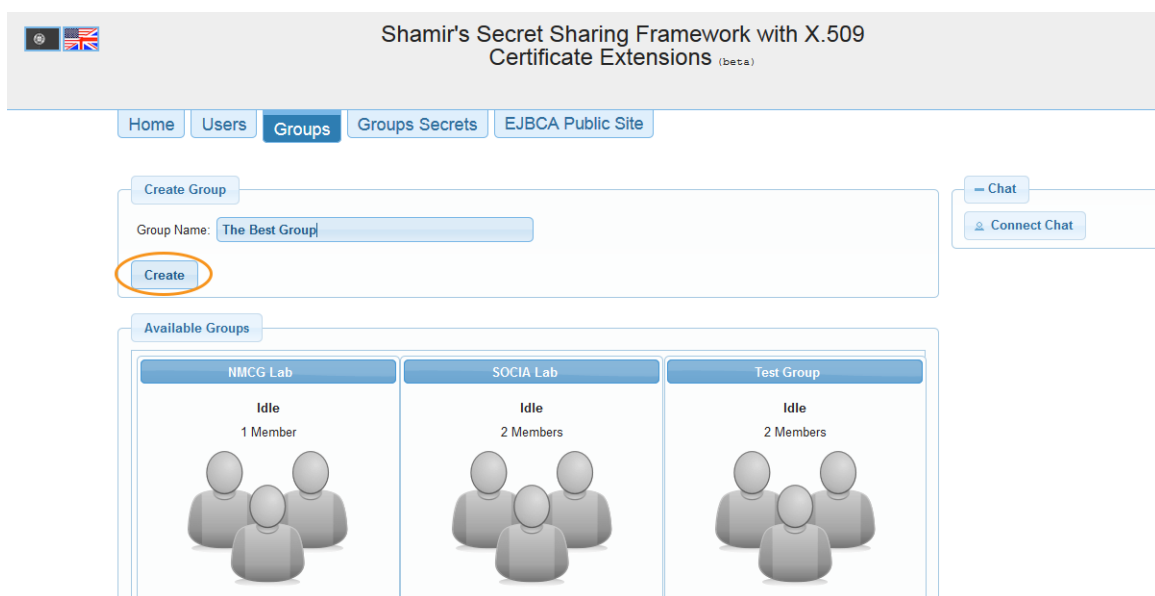


Figure 5.10: Web application group create



Figure 5.11: Web application group join

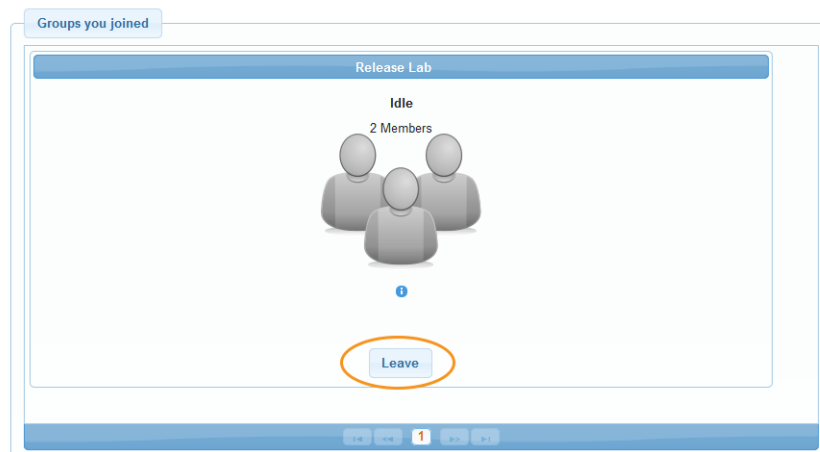


Figure 5.12: Web application group leave

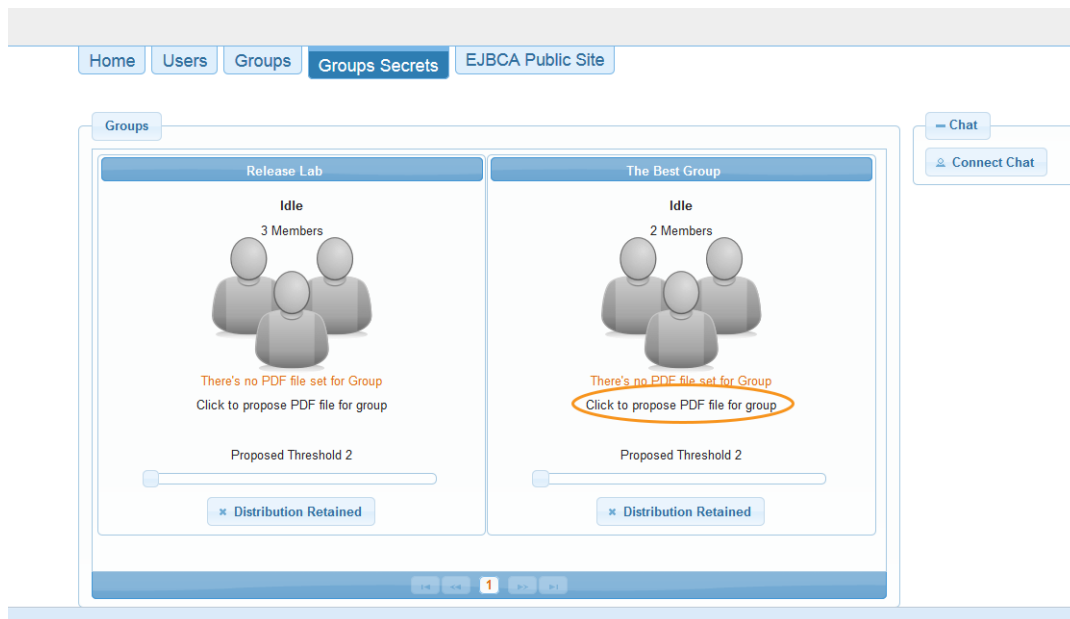


Figure 5.13: Web application propose PDF

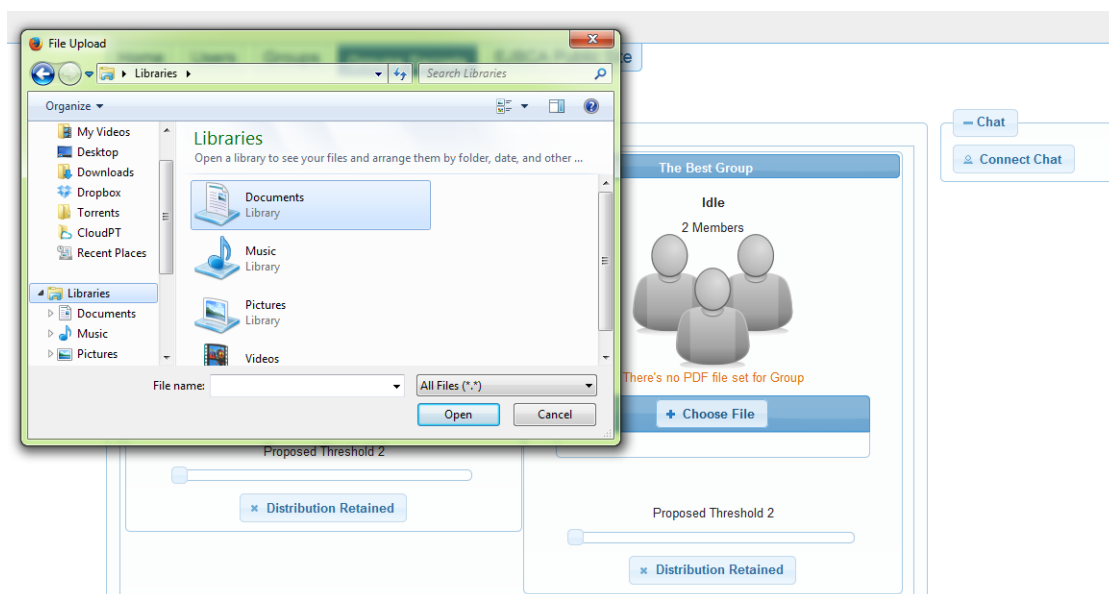


Figure 5.14: Web application get PDF file dialog

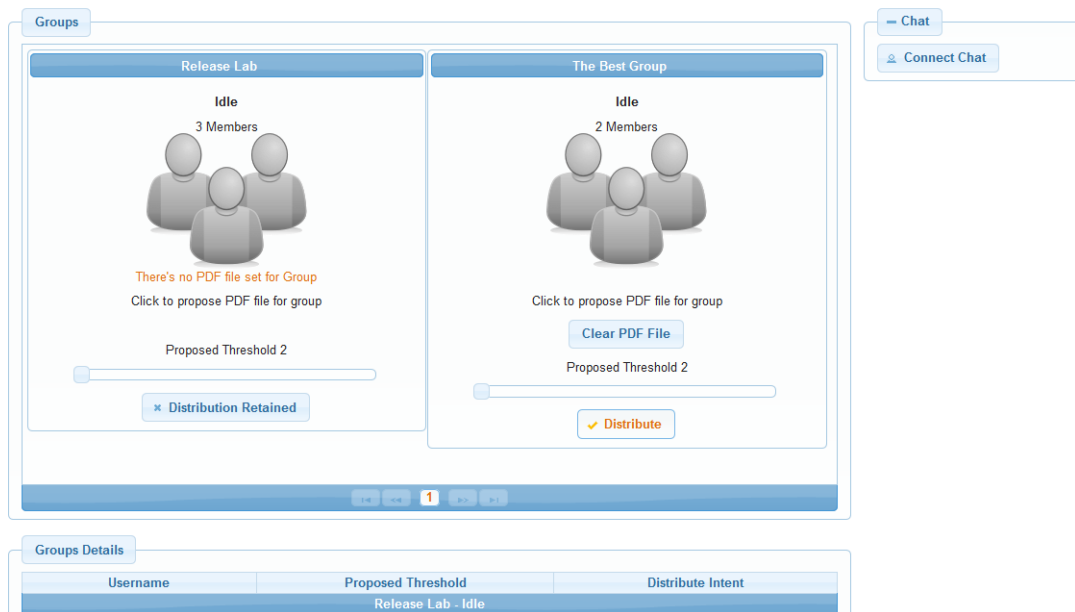


Figure 5.15: Web application secret distribute intent

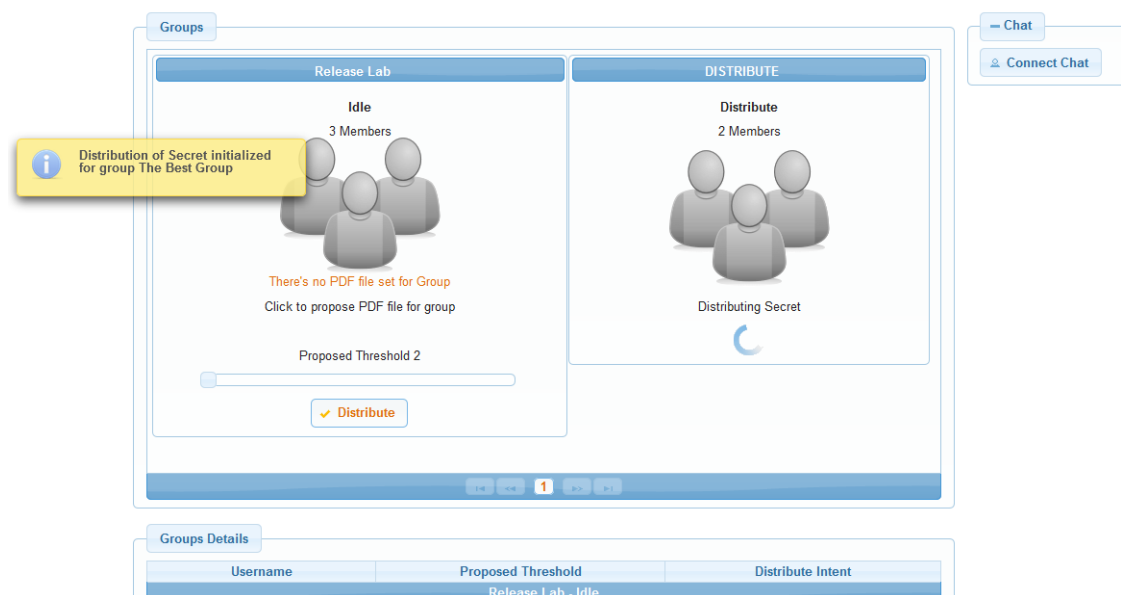


Figure 5.16: Web application secret distribution initiated



Figure 5.17: Web application secret recover intent

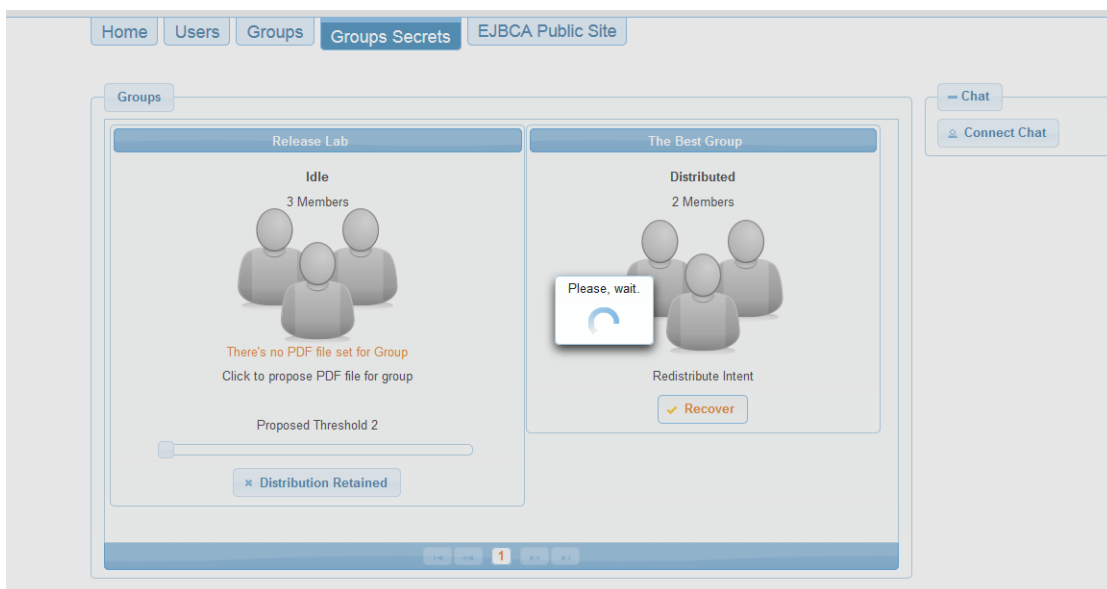


Figure 5.18: Web application secret recover initiated

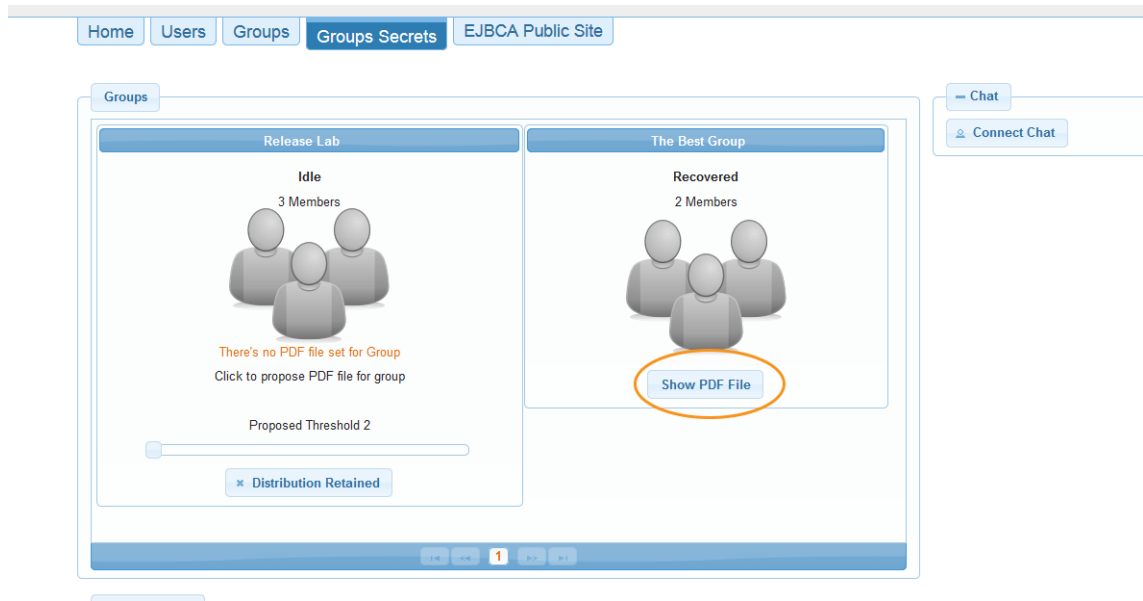


Figure 5.19: Web application show PDF document

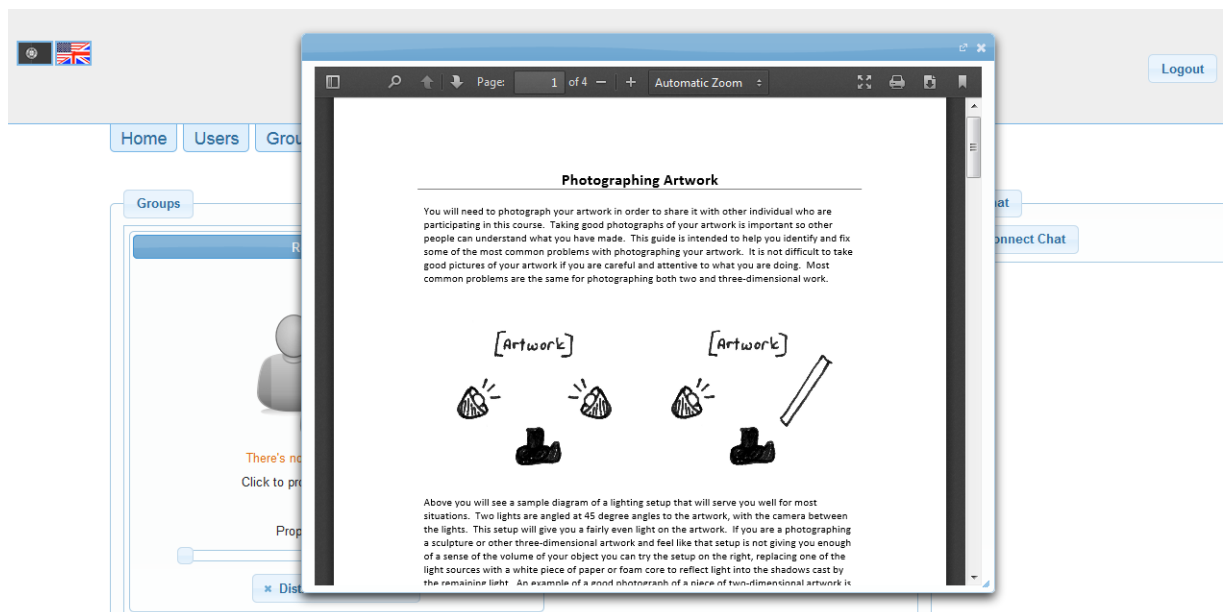


Figure 5.20: Web application recovered PDF document





# Chapter 6

## Security Analysis

### Contents

<b>5.1</b>	<b>EJBCA Deployment Process</b>	<b>35</b>
5.1.1	Securing MySQL RDBMS	35
5.1.2	Securing JBoss Server	36
5.1.3	EJBCA Installation	36
5.1.4	CA Hierarchy Implementation	38
<b>5.2</b>	<b>Web Application Technologies</b>	<b>41</b>
5.2.1	Securing GlassFish Server	41
5.2.2	Framework on the Web	42
5.2.3	Web Application Workflow	43

## 6.1 Optimization and Obsfuscation

The Web Service RA application goes through a process of optimization and obfuscation. These improve overall Java byte-code performance and difficult the task of reverse-engineering.

### 6.1.1 ProGuard

ProGuard is a Java class file shrinker, optimizer, obfuscator, and pre-verifier. The shrinking step detects and removes unused classes, fields, methods, and attributes. The optimization step analyzes and optimizes the byte-code of the methods. The obfuscation step renames the remaining classes, fields, and methods using short meaningless names.

These steps make the code base smaller, more efficient, and harder to reverse-engineer.

Each of these steps is optional. For instance, ProGuard can also be used to just list dead code in an application, or to pre-verify class files for efficient use in Java 6.

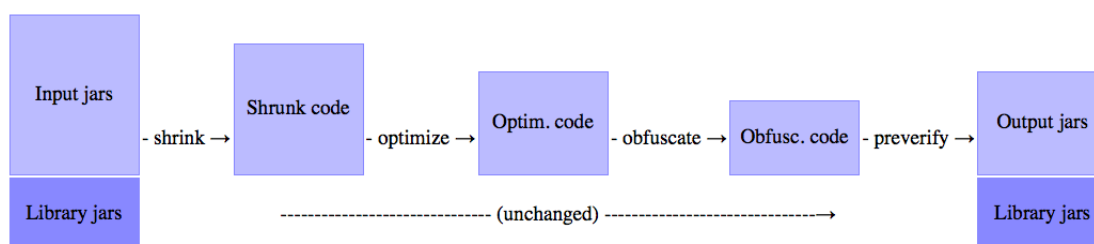


Figure 6.1: Proguard optimization-obfuscation process

ProGuard initially reads the input jars (or wars, ears, zips, or directories). It then subsequently shrinks, optimizes, obfuscates, and pre-verifies them. ProGuard can optionally perform multiple optimization passes. ProGuard writes the processed results to one or more output jars (or wars, ears, zips, or directories). The input may contain resource files, whose names and contents can optionally be updated to reflect the obfuscated class names.

ProGuard requires the library jars (or wars, ears, zips, or directories) of the input jars to be specified. These are essentially the libraries that would be needed to compile the code. ProGuard uses them to reconstruct the class dependencies that are necessary for proper processing. The library jars themselves always remain unchanged.

## 6.2 SSL-TLS Audit

In this section we will provide a SSL-TLS assessment of correct installation and functioning. There are several online tools for security audits of SSL-TLS installation, we used Qualys<sup>®</sup> (qua) SSL labs online service to perform a deep analysis of the configuration of our web servers. We choose Qualys<sup>®</sup> (qua) since its used by several high profile companies such as: HP<sup>®</sup>, Microsoft<sup>®</sup>, CISCO<sup>®</sup>, eBay<sup>®</sup>, Adobe<sup>®</sup>, Oracle<sup>®</sup>, etc.

### 6.2.1 Qualys SSL-TLS Server Rating Guide

#### Methodology Overview

The SSL-TLS server rating approach consists of three steps:

1. First look at a certificate to verify that it is valid and trusted. A server that fails this step is always assigned a zero score <sup>1</sup>.
2. Inspect server configuration in three categories:
  - a. Protocol support
  - b. Key exchange support
  - c. Cipher support
3. Final score (a number between 0 and 100) is a combination of the scores achieved in the individual categories. A zero score in any category forces the total score to zero.

---

<sup>1</sup>Since we have our own CA and use a self-signed root certificate our certificates are not trusted by other CA's thus, we get a score of 0. We don't consider this an issue or a misconfiguration since its expected.

Because small differences between configurations are sometimes less important, Qualys<sup>®</sup> also assign letter grades to servers. Letter grades are generally more useful, as its instantly clear that a server given an A is well configured, opposing to server with a letter F. Table 6.1 shows how a numerical score is translated into a letter grade.

Letter grade translation	
<i>Numerical Score</i>	<i>Grade</i>
score $\geq 80$	A
score $\geq 65$	B
score $\geq 50$	C
score $\geq 35$	D
score $\geq 20$	E
score $< 20$	F

Table 6.1: Letter grade translation from numerical score

## Certificate Inspection

Qualys<sup>®</sup> SSL-TLS check considers the following certificates issues with a result of a zero score:

- Domain name mismatch;
- Certificate not yet valid;
- Certificate expired;
- Use of a revoked certificate;
- Use of a self-signed certificate <sup>2</sup>;
- Use of a certificate that isn't trusted (unknown CA or some other validation error) <sup>3</sup>;

SSL is a complex hybrid protocol with support for many features across several phases of operation. To account for the complexity, Qualys<sup>®</sup> rate the configuration of an SSL server in three categories, as displayed in Table 6.2. The final score is a combination of the scores in the individual categories, as described in the “Methodology Overview” 6.2.1 section.

Criteria categories	
<i>Category</i>	<i>Score</i>
Protocol support	30%
Key exchange	30%
Cipher strength	40%

Table 6.2: Score percentage of each category

<sup>2</sup>Our server configuration falls in this category, since we use a self-signed root certificate.

<sup>3</sup>Our server configuration falls in this category, since we have our own CA and it isn't trusted by any other CA.

## 6.2.2 JBoss 5.1 SSL-TLS Audit

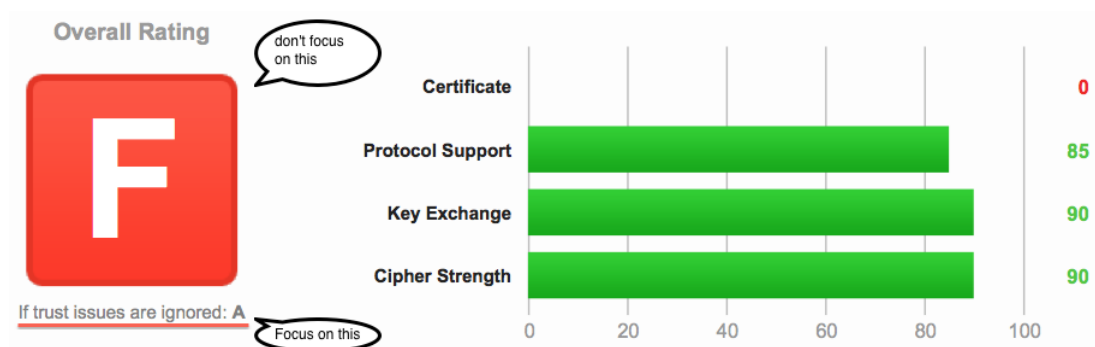



Figure 6.2: JBoss 5.1 SSL-TLS score


	Protocols	
	TLS 1.2	No
	TLS 1.1	No
	TLS 1.0	Yes
	SSL 3	Yes
	SSL 2	No
	Cipher Suites (sorted by strength; the server has no preference)	
	TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)	128
	TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	168
	TLS_RSA_WITH_AES_256_CBC_SHA (0x35)	256

Figure 6.3: JBoss 5.1 SSL-TLS supported protocols



Protocol Details	
Secure Renegotiation	Supported
Secure Client-Initiated Renegotiation	No
Insecure Client-Initiated Renegotiation	No
BEAST attack	Not mitigated server-side ( <a href="#">more info</a> ) SSL 3: 0x2f, TLS 1.0: 0x2f
TLS compression	No
RC4	No
Forward Secrecy	No NOT DESIRABLE ( <a href="#">more info</a> )
Next Protocol Negotiation	No
Session resumption	Unknown
Session tickets	No
OCSP stapling	No
Strict Transport Security	Unknown
Long handshake intolerance	No
TLS extension intolerance	No
TLS version intolerance	TLS 2.98
SSL 2 handshake compatibility	Yes

Figure 6.4: JBoss 5.1 SSL-TLS supported protocol details



Certification Paths	
Path #1: Not trusted (path does not chain to a trusted anchor)	
1	Sent by server EJBCA Server SHA1: be4b42a71c7786fb8e862406e3456047db5ab124 RSA 2048 bits / SHA256withRSA
2	Sent by server Servidor CA SHA1: dc67c57f6f6f45d85659ce54beff9ca8eafa5a4b RSA 2048 bits / SHA256withRSA
3	Sent by server Not in trust store Raiz CA SHA1: f6fa896f8423d3842d52c7ad0f1a128aa85b2994 RSA 2048 bits / SHA256withRSA

Figure 6.5: JBoss 5.1 SSL-TLS certification path

### 6.2.3 Glassfish 3.1.2 SSL-TLS Audit



Figure 6.6: Glassfish 3.1.2 SSL-TLS score


	Protocols	
	TLS 1.2	No
	TLS 1.1	No
	TLS 1.0	Yes
	SSL 3	No
	SSL 2	No
	Cipher Suites (sorted by strength; the server has no preference)	
	TLS_RSA_WITH_AES_128_CBC_SHA (0x2f)	128
	TLS_RSA_WITH_3DES_EDE_CBC_SHA (0xa)	168
	TLS_RSA_WITH_AES_256_CBC_SHA (0x35)	256

Figure 6.7: Glassfish 3.1.2 SSL-TLS supported protocols



Protocol Details	
Secure Renegotiation	Supported
Secure Client-Initiated Renegotiation	No
Insecure Client-Initiated Renegotiation	No
BEAST attack	Not mitigated server-side ( <a href="#">more info</a> ) TLS 1.0: 0x2f
TLS compression	No
RC4	No
Forward Secrecy	No NOT DESIRABLE ( <a href="#">more info</a> )
Next Protocol Negotiation	No
Session resumption	Unknown
Session tickets	No
OCSP stapling	No
Strict Transport Security	Unknown
Long handshake intolerance	No
TLS extension intolerance	No
TLS version intolerance	TLS 2.98
SSL 2 handshake compatibility	Yes

Figure 6.8: Glassfish 3.1.2 SSL-TLS supported protocol details



Certification Paths	
Path #1: Not trusted (path does not chain to a trusted anchor)	
1	<div>Sent by server</div> <div>Secret Sharing Framework Server</div> <div>SHA1: 0a06750bb391cea18e2cd4c37a39b72efa301c24</div> <div>RSA 2048 bits / SHA256withRSA</div>
2	<div>Sent by server</div> <div>Servidor CA</div> <div>SHA1: dc67c57f6f45d85659ce54beff9ca8eafa5a4b</div> <div>RSA 2048 bits / SHA256withRSA</div>
3	<div>Sent by server</div> <div>Not in trust store</div> <div>Raiz CA</div> <div>SHA1: f6fa896f8423d3842d52c7ad0f1a128aa85b2994</div> <div>RSA 2048 bits / SHA256withRSA</div>

Figure 6.9: Glassfish 3.1.2 SSL-TLS certification path

## 6.2.4 SSL-TLS Audits on other Organizations

### SSL-TLS Audit on CGD (Portuguese Bank Caixa Geral de Depósitos)

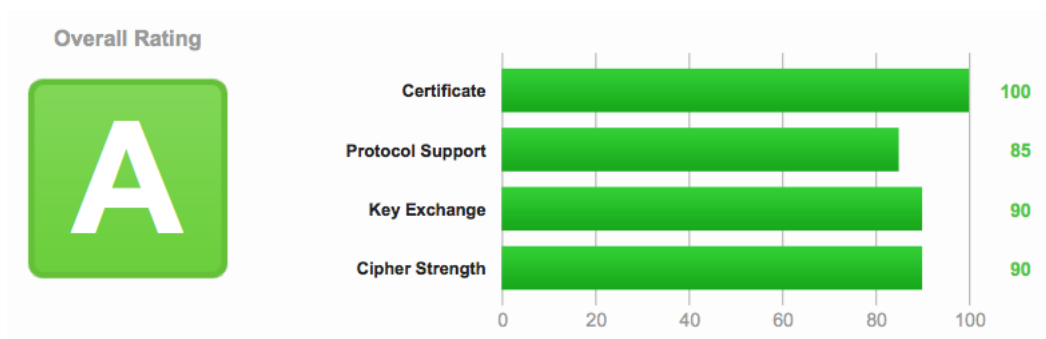


Figure 6.10: Caixa Geral de Depósitos SSL-TLS score

### SSL-TLS Audit on UBI (Portuguese University of Beira Interior)

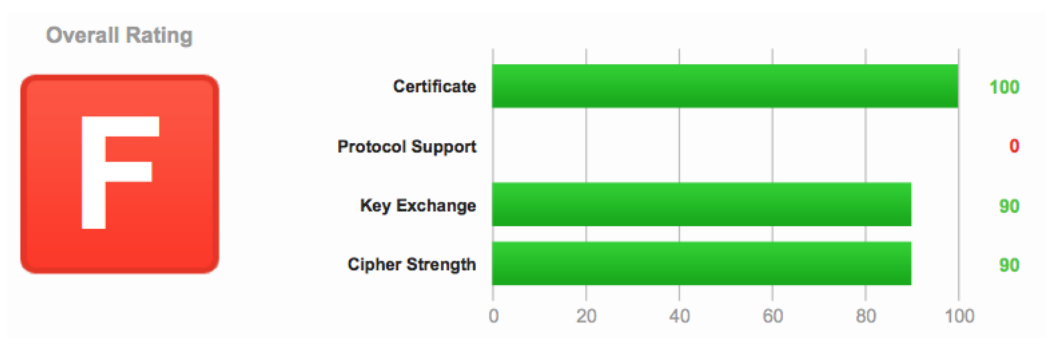


Figure 6.11: University of Beira Interior SSL-TLS score

### SSL-TLS Audit on Qualys-SSL Labs (Self-Server Test)

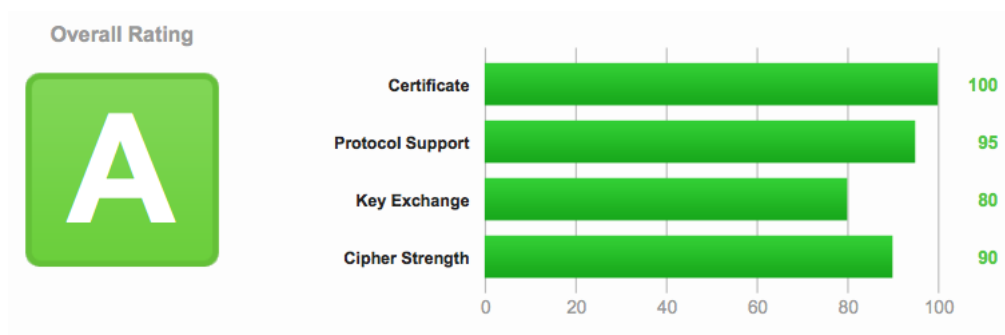


Figure 6.12: Qualys-SSL Labs SSL-TLS score



### 6.2.5 Conclusions

Deductively from the following scores we get to the conclusion that the securest server is from Qualys® it self, or that it is the most well configured according to what is known about *Common Vulnerabilities and Exposures* (CVE).

UBI (Universidade da Beira Interior) receive the lowest score since they support the, now considered, insecure protocol SSL 2.0. Hence by Qualys® risk assessment methodologies they received a 0 score in protocol support, also Browser Exploit Against SSL/TLS (BEAST) (MS13),(SF13) isn't mitigated server side.

We can also conclude that our servers are “almost” as secure as CGD (Caixa Geral de Depósitos), if Browser Exploit Against SSL/TLS (BEAST) was mitigated server side and *forward secrecy*<sup>4</sup> protocols where supported, they would probably be.

The previous “recipe” to workaround BEAST had been to enforce RC4 ciphers when TLS 1.0 was used, this as become *flawed* (MS13),(SF13) as the “cure” can be worse than the “disease”. The only “cure” is to advance to more recent protocols TLS 1.1 or TLS.1.2.

Even with the exclusion of *weak* ciphers from Glassfish and JBoss they will only still support SSL 3.0 and TLS 1.0. these are architectural limitations that we cannot workaround, these are “Community Editions” or “Open Source”.

## 6.3 Attack Trees

Attack trees provide a way to model threats against computer systems in a formal and methodical way (Sch). The first element to be identified in the model is the goal of the attack. This element is going to be the root element of the tree. Then, continue to identify the different ways of achieving that goal. Each different way will be a leaf node of the tree. The Figure 6.13 illustrates an attack tree that identifies some ways that can be used to open a safe.

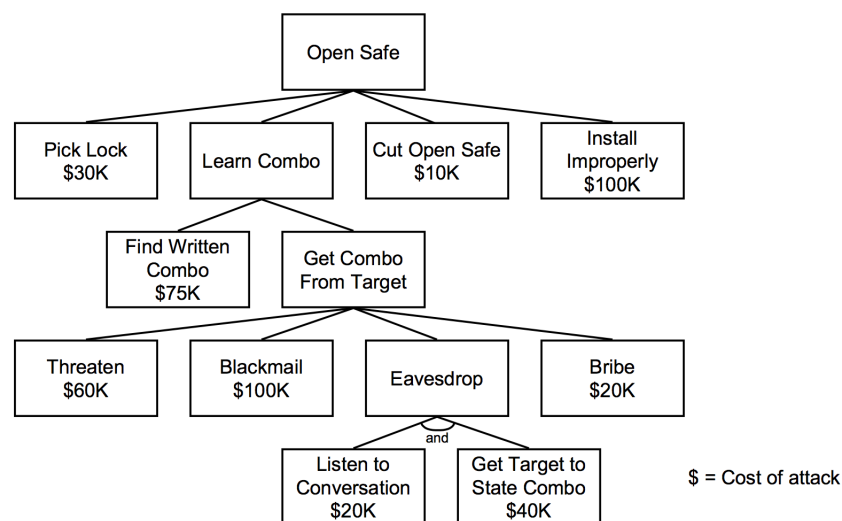


Figure 6.13: Attack Tree - Open Safe. Adapted from (Sch).

<sup>4</sup>All Diffie-Hellman protocol variants present this property.

In attack trees it's also possible to estimate costs if a given goal is achieved, or calculate how probable it's to achieve such goal. In order to estimate such values we must attach to each leaf node of the tree the cost (or probability) of the attack. As we can see in Figure 6.13 the estimated cheapest cost of attack is \$10K.

In the mentioned example it was used cost, but other values could be attached to the nodes as well. For instance, one may attach to each leaf node of the tree: probabilities, risk categories (low, moderate, high), possibility (possible, impossible), skills.

An attack tree can be reused, that is, it is possible to use an attack tree as a node of another one. This is very handy when you have a large system and there are several people analyzing its security, and when there is an attack that can be used in several attack trees of a system.

### 6.3.1 Models of Attacks on the Framework

Throughout this section we will present and discuss the attack trees we've created during the security analysis. These attack trees will expose several threats that can be perpetrated to each of the following goals we identified:

- Forge certificate request;
- Steal Certificate;
- Collect user data.

## Forge Certificate Request

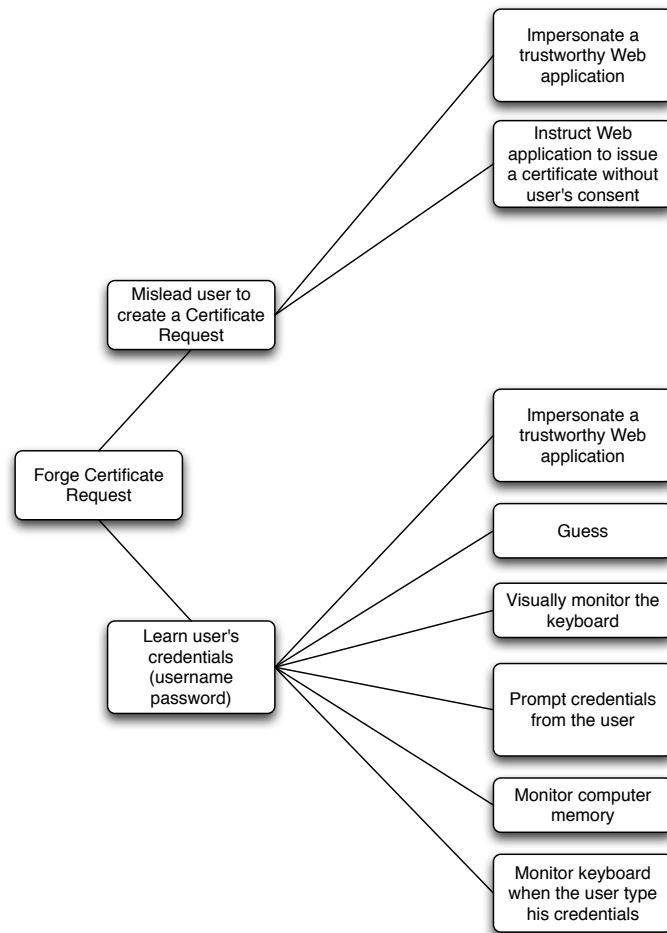


Figure 6.14: Attack Tree to forge a certificate request

## Steal Certificate Request

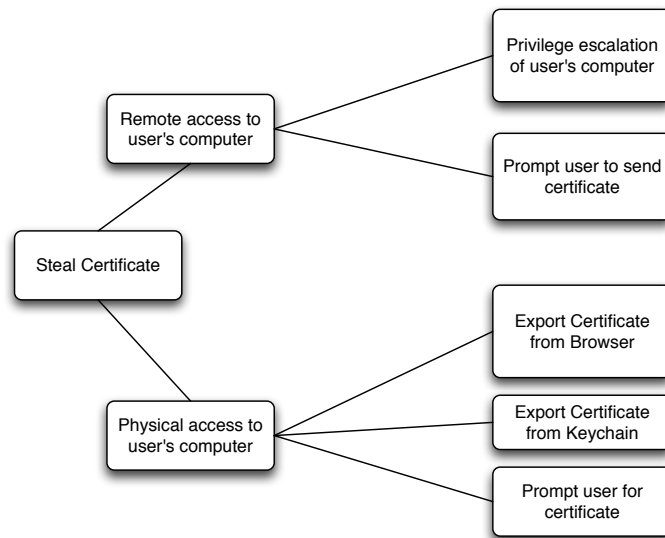


Figure 6.15: Attack Tree to steal a certificate

## 6.4 Conclusions

As stated in (Sch): “Security is not a product - it’s a process”. In that sense, the work performed in the security analysis of the framework is just a first step towards a full comprehension of risks and vulnerabilities.

The security analysis using attack trees to model goals and threats is far from complete. These are just initial steps towards a global understanding of the system.

The framework that we developed is destined to be used by web-based systems, thus we must ensure security in the four fronts: web client, data transport, web server, and operating system. In that sense, we should not only perform an more detailed audit to the used applications source code and to its execution, but to the surrounding system.

From such analysis we can then understand how the system security is affected by the different applications that comprise our framework, and vice-versa, in order to create a higher protection for users and framework.



# Chapter 7

## Conclusions and Further Work

### Contents

---

<b>6.1 Optimization and Obsfuscation . . . . .</b>	<b>53</b>
6.1.1 ProGuard . . . . .	53
<b>6.2 SSL-TLS Audit . . . . .</b>	<b>54</b>
6.2.1 Qualys SSL-TLS Server Rating Guide . . . . .	54
6.2.2 JBoss 5.1 SSL-TLS Audit . . . . .	56
6.2.3 Glassfish 3.1.2 SSL-TLS Audit . . . . .	58
6.2.4 SSL-TLS Audits on other Organizations . . . . .	60
6.2.5 Conclusions . . . . .	61
<b>6.3 Attack Trees . . . . .</b>	<b>61</b>
6.3.1 Models of Attacks on the Framework . . . . .	62
<b>6.4 Conclusions . . . . .</b>	<b>65</b>

---

### 7.1 Conclusions

The premise of this thesis was that, it was plausible that a PKI could be build with support for *Threshold Cryptography* by embedding a custom certificate extension in the *digital certificates*.

To achieve this goal, we explored the existing literature, as mentioned in the state of art chapter 2, and concluded that the more suitable approach for our development would be Shamir's SSS.

Forward, in chapter 3 we consolidated the premise that it is plausible to define/build a custom certificate extension to carry unique information from RFC-5280 section 4.2.

The 4th chapter, resumes the architectural model of the framework. We start by explaining the communication process between the CA and Web Application and the importance of being a holistic model.

Here we clarify the secret distribution as well as the secret recovery, presenting the respective algorithms.

Next we presented procedures for the deployment of our "proof of concept" framework, with all of its components CA, Web service RA and Web application. Without neglecting the security procedures.

In the security analysis of our framework. We've shown how to perform some optimization and obsfuscation processes on the Web service RA application. An SSL security assessment tool that analyses correct installation as well as some flaws was used and it's methodologies explained.

We've ended our security analysis with attack trees for a formal visualization of attack paths to our framework.

Overall, it's plausible to state that the aim of the thesis was achieved. The platform in fact uses *custom certificate extensions* in order to support Shamir's SSS.

## 7.2 Future Work

Regarding the future work, this Secret Sharing platform has a wide range of possibilities for improvements.

Threshold Crypto-Systems have an high applicability in a more social world where people are *always connected* as Secret Sharing is inherently a group collaborative action as it requires individuals working together in a coordinated fashion, towards a common goal, namely sharing and recovering of secrets.

Future work should also consider testing more thoroughly our framework with many concurrent users.



# Appendix A

## Annexes

### Contents

7.1	Conclusions . . . . .	67
7.2	Future Work . . . . .	68

## A.1 EJBCA Configuration Property Files

We will not present the whole file transcripts but only the relevant properties we've configured <sup>1</sup>.

### A.1.1 Certificate Store Configuration

```
certstore.enabled=true
```

### A.1.2 CRL Store Configuration

```
crlstore.enabled=true
```

### A.1.3 Database Configuration

```
database.name=mysql
database.url=jdbc:mysql://127.0.0.1:3306/ejbcaadb?characterEncoding=UTF-8
database.username=ejbca-user
database.password=*****
```

### A.1.4 EJBCA Configuration

```
ca.keystorepass=:*****
intresources.secondarylanguage=PT
healthcheck.amountfreemem=32
ejbca.passwordlogrounds=16 2
```

### A.1.5 OCSP Configuration

```
ocsp.enabled=true
ocsp.signaturealgorithm=SHA256WithRSA;SHA1WithRSA
```

<sup>1</sup>All passwords follow the specifications referred on 3.1.4

<sup>2</sup>To better protect from off-line brute force attacks of passwords on a compromised database, the computationally expensive *BCrypt* algorithm can be configured to use higher log-rounds values that will increase computational cost. Values can be between 1,...,31. "A decent value for high security is 8". According to EJBCA sample file.

## A.1.6 EJBCA Install Configuration

```
ca.name=AdminCA1
ca.dn=CN=AdminCA1,O=EJBCA Sample,C=SE
ca.tokenpassword=soft
ca.tokenpassword=null
ca.keyspec=2048
ca.keytype=RSA
ca.signaturealgorithm=SHA256WithRSA
ca.validity=13
ca.policy=null
```

## A.1.7 Web Configuration

```
java.trustpassword=*****
superadmin.cn=SuperAdmin
superadmin.dn=CN=$superadmin.cn httpsserver.password=*****
httpsserver.hostname=www.secret-sharing.tk
httpsserver.dn=CN=$httpsserver.hostname,O=EJBCA Sample,C=SE
web.availablelanguages=EN,PT
log.maxqueryrowcount=100
```

---

<sup>3</sup>As this certificate is only being used to deploy EJBCA for the first time, so it only has 1 day validity. We will build our CA hierarchy with other certificates.

## A.1.8 Certificate Extensions Configuration

```
id1.oid=2.999
id1.classpath=org.ejbca.core.model.ca.certextensions.BasicCertificateExtension
id1.displayname=SSS-EXTENSION
id1.used=true
id1.translatable=false
id1.critical=false
id1.property.encoding=DERUTF8STRING
id1.property.dynamic=true
id1.property.value=null
```

## A.2 Diagrams

In our framework development we use an *Unified Modeling Language* (UML) modeling approach. The following figure A.1 highlights the used modeling diagrams.

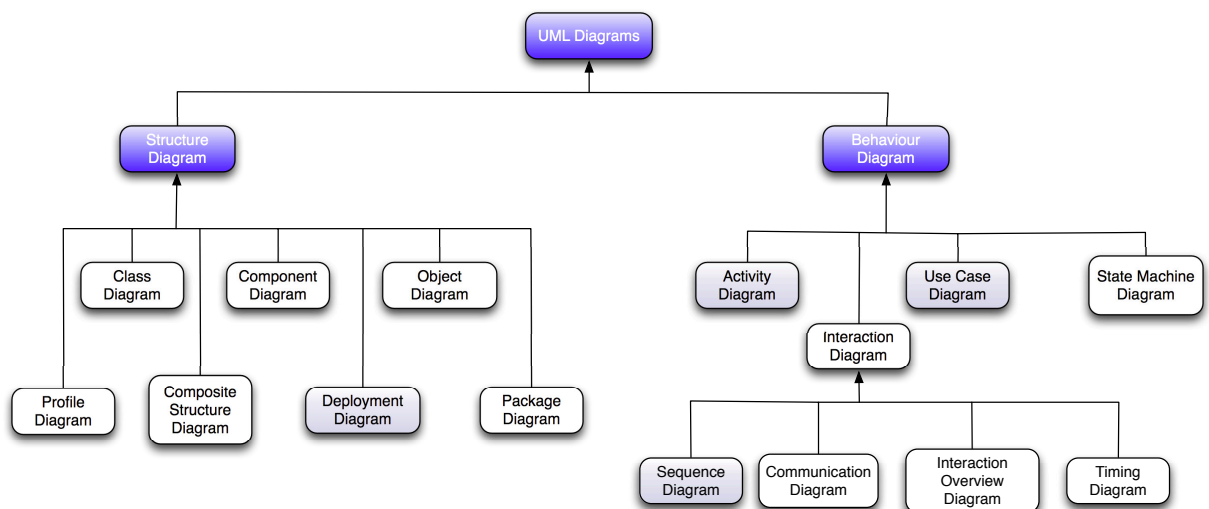


Figure A.1: UML diagrams overview.

## A.2.1 Use Case Diagrams

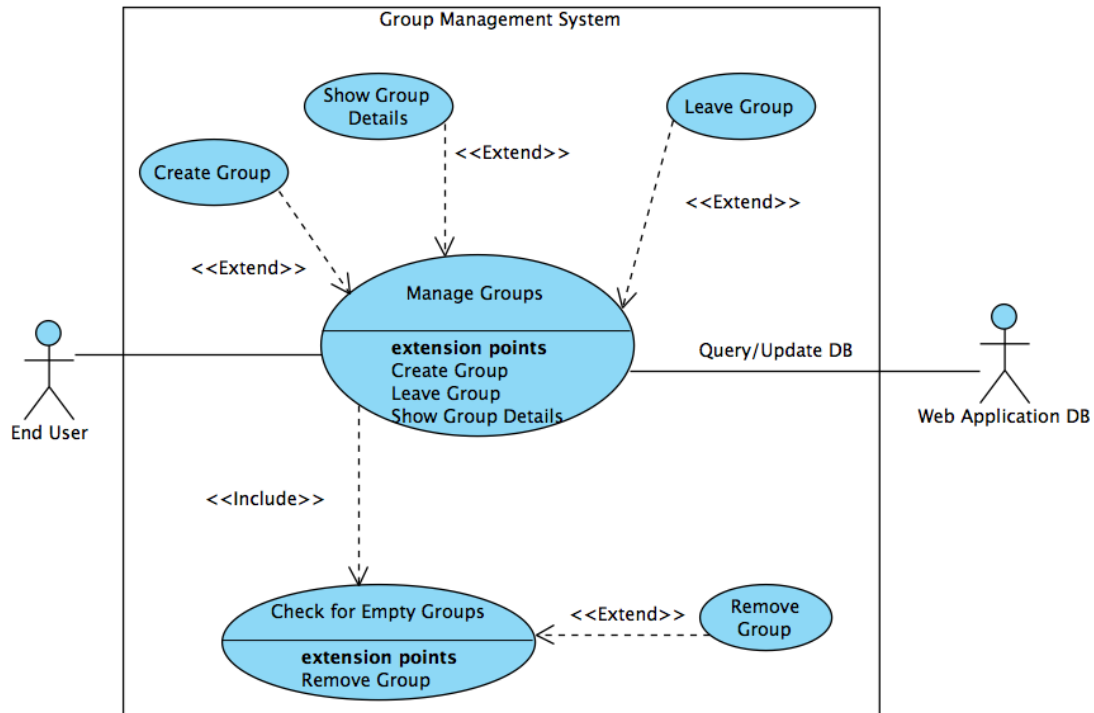


Figure A.2: Use case diagram of group management system

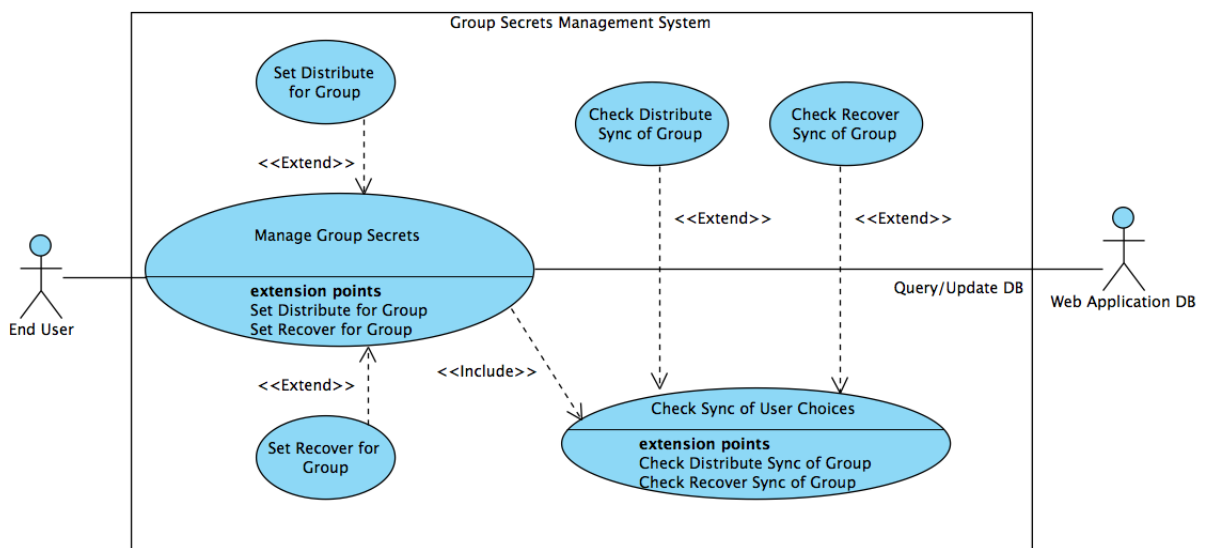


Figure A.3: Use case diagram of group secrets management system

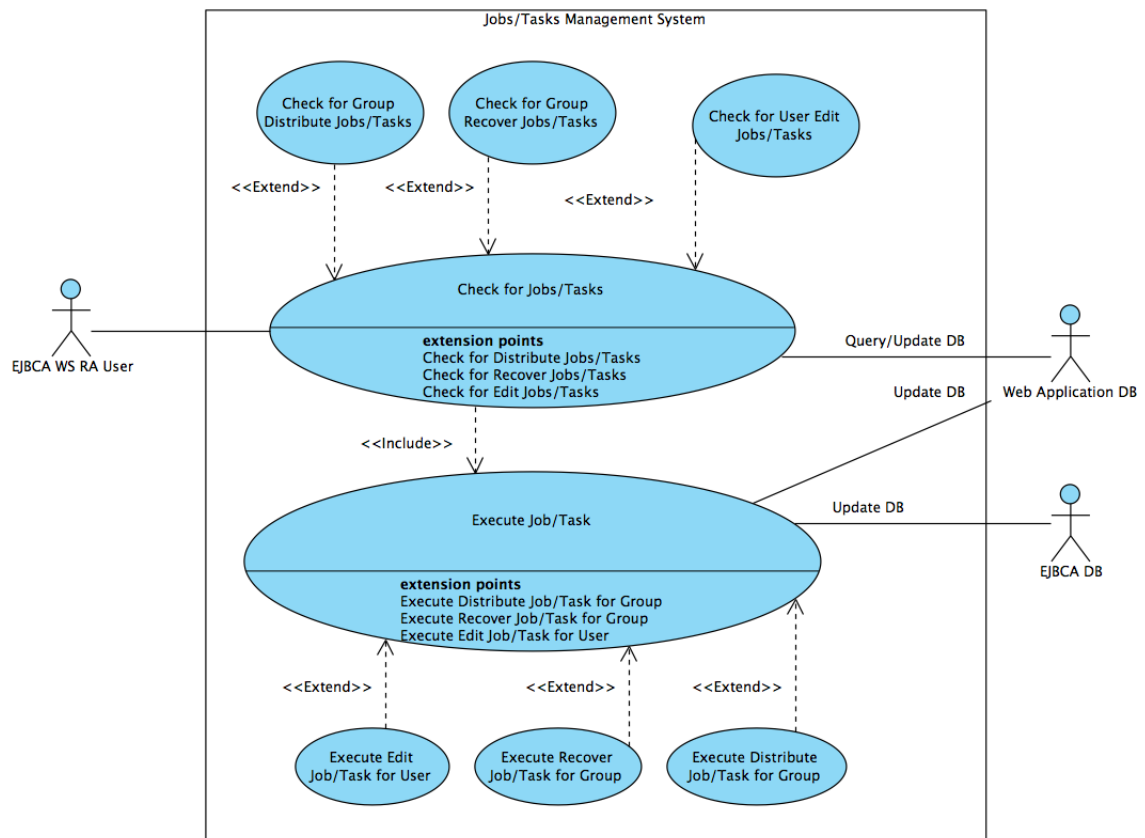


Figure A.4: Use case diagram of the jobs/tasks management system

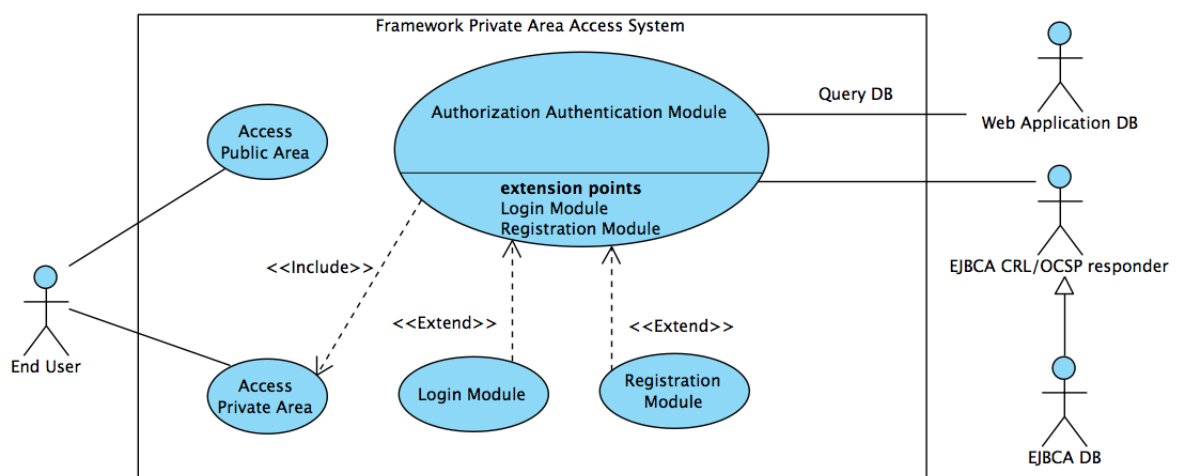


Figure A.5: Use case diagram for access to the private area of the Web application

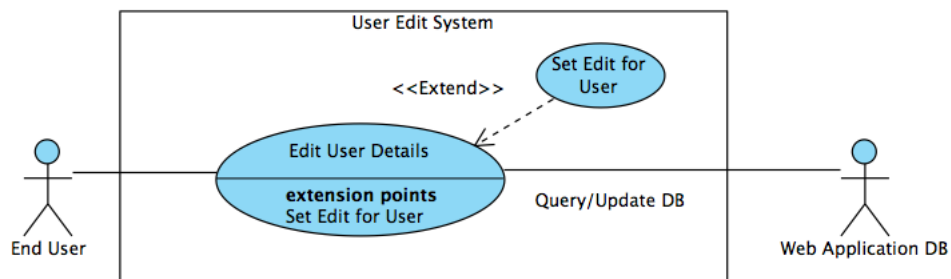


Figure A.6: Use case diagram of the user edit system

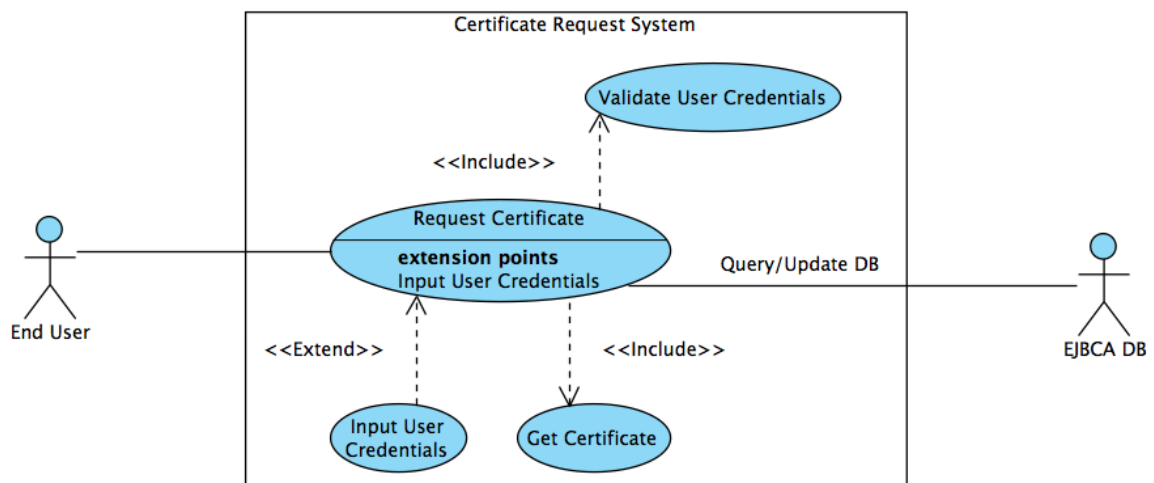


Figure A.7: Use case diagram of the certificate request system

## A.2.2 Sequence Diagrams

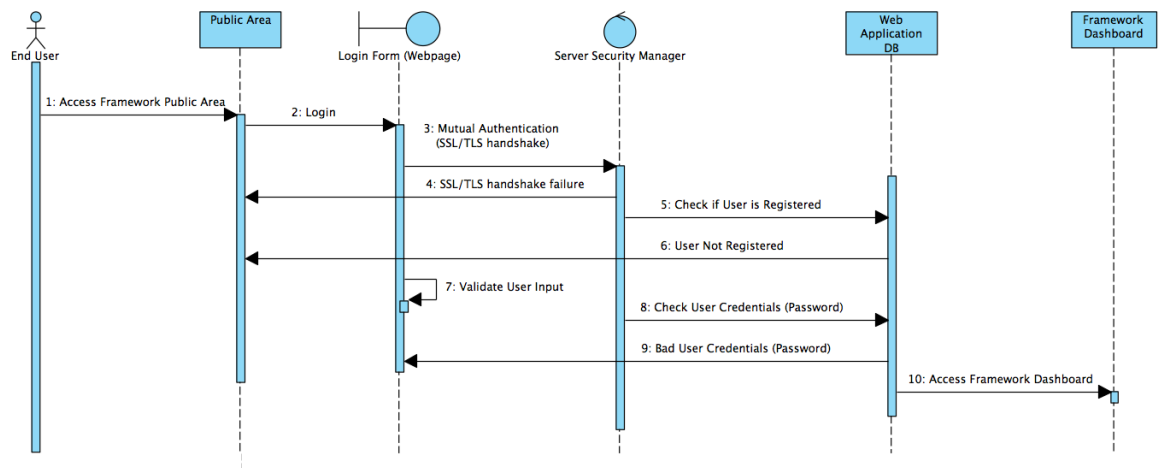


Figure A.8: Sequence diagram for user login

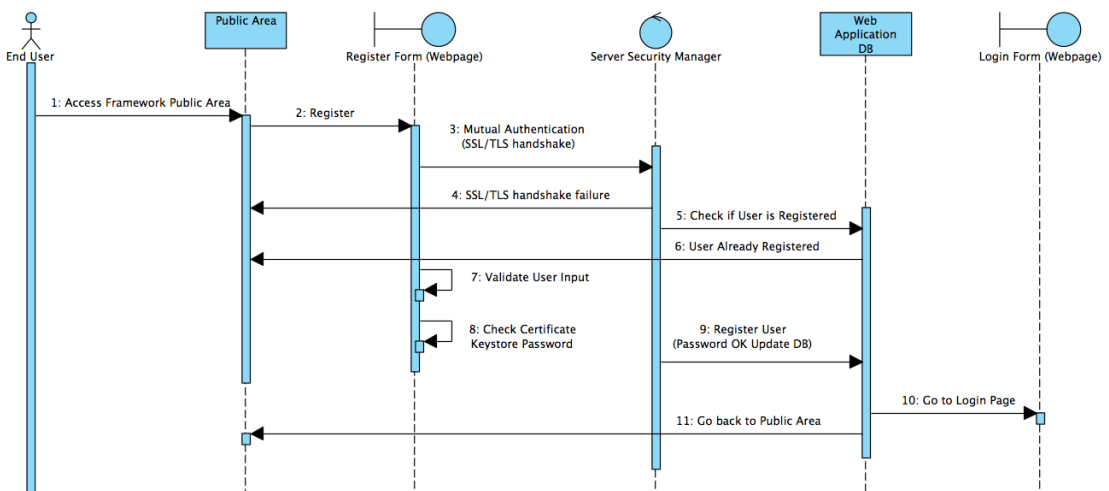


Figure A.9: Sequence diagram for user register

### A.2.3 Activity Diagrams

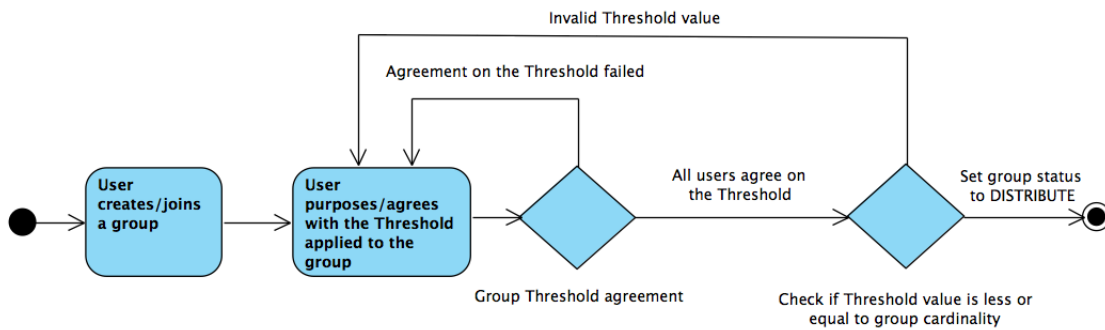


Figure A.10: Activity diagram of the distribute process

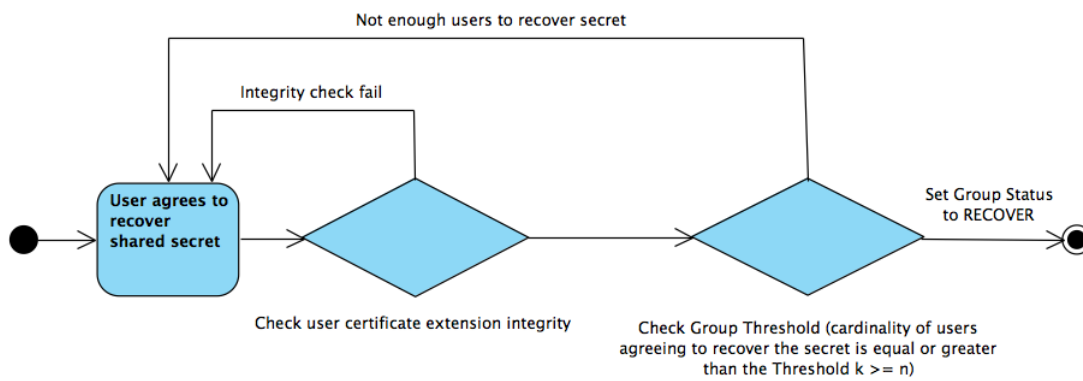


Figure A.11: Activity diagram of the recover process



#### A.2.4 Deployment Diagrams

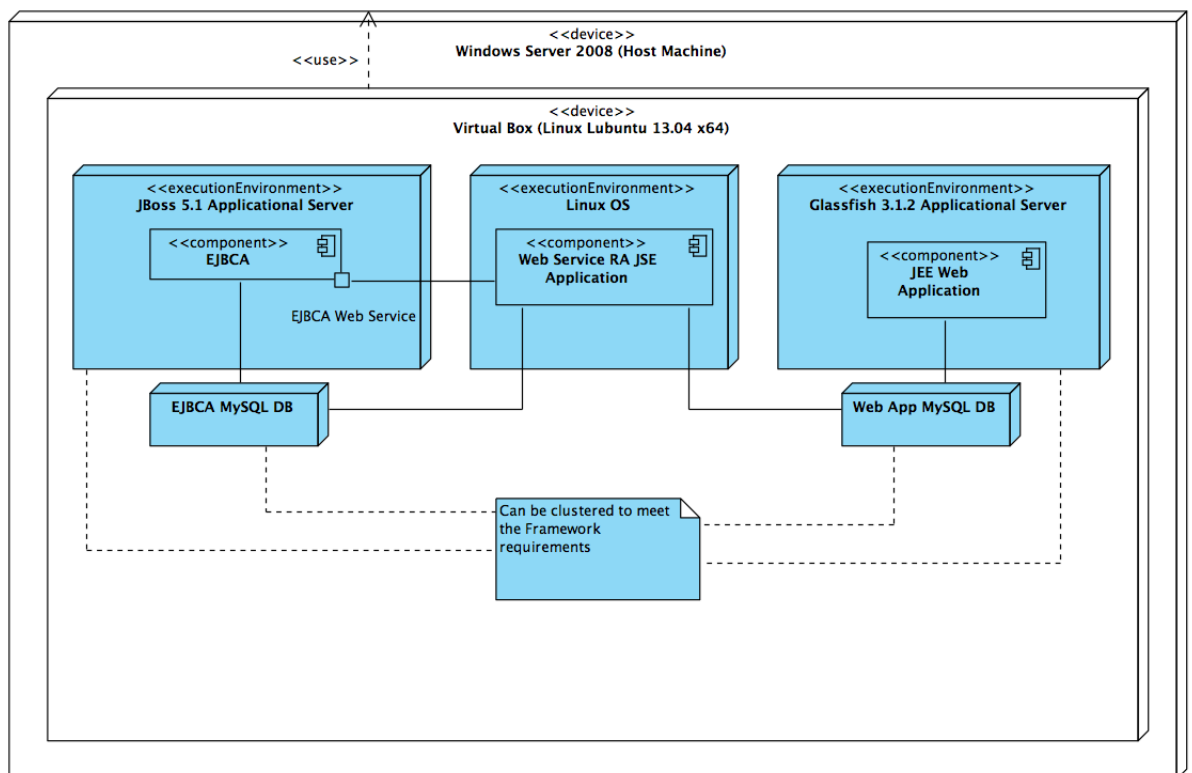


Figure A.12: Deployment diagram of the framework

## A.2.5 DataBase Entity Relationship Diagram

This section denotes the ERD used by the Web application and the Web service Java application.

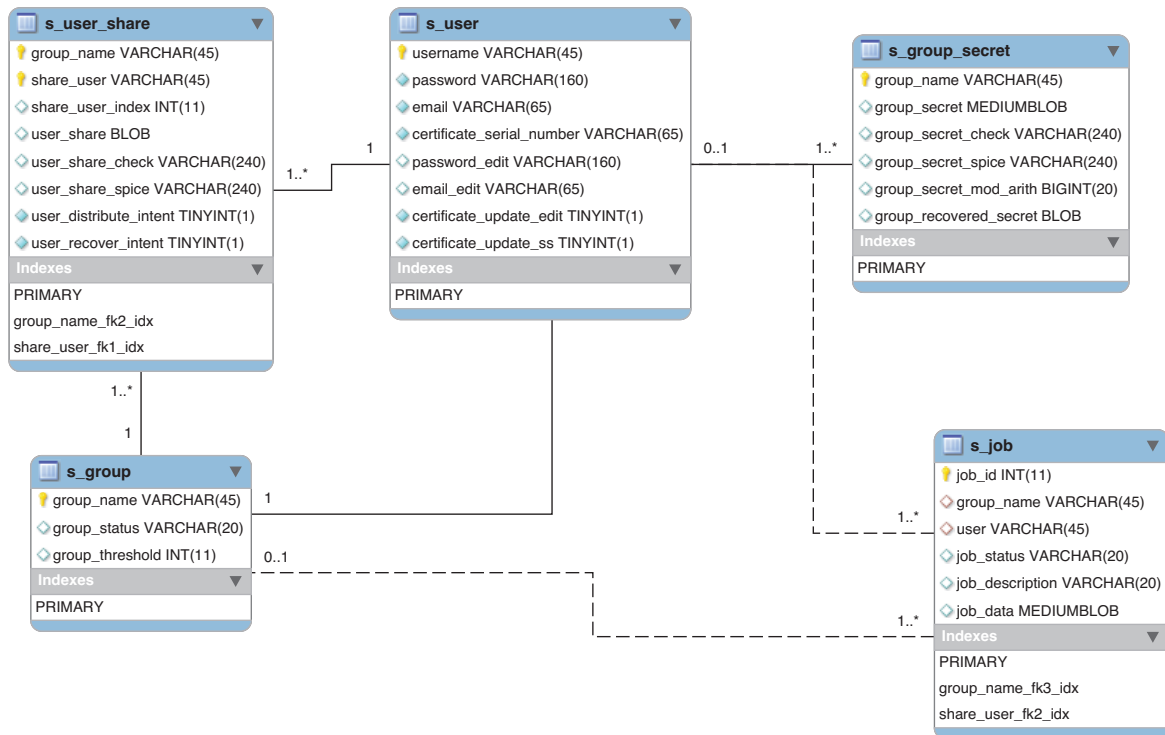


Figure A.13: Entity Relationship Diagram of the Web application database

# Bibliography

- [BB84] C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, page 175, 1984. 11
- [Ben87] J. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Advances in Cryptology — CRYPTO’ 86*, pages 251–260, 1987. 9, 10
- [Ben96] J. Benaloh. *Variable Secret-Ballot Elections*. PhD thesis, Yale, 1996. 10
- [Beu89] Beutelspacher. How to say “no”. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1989. 8
- [BL90] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions, 1990. 8
- [Bla79] G.R. Blakley. Safeguarding cryptographic keys. In *AFIPS*, page 313. IEEE Computer Society, 1979. 5, 7, 9, 11
- [BM85] G. R. Blakley and Catherine Meadows. Security of ramp schemes. In G. R. Blakley and D. C. Chaum, editors, *Proceedings in CRYPTO 84*, pages 242–268. Springer, 1985. Lecture Notes in Computer Science No. 196. 9, 10
- [Bra] 14
- [BS97] C. Blundo and D.R. Stinson. Anonymous secret sharing schemes. *Discrete Applied Mathematics*, (96), 1997. 9
- [BSGV93] Blundo, De Santis, Gargano, and Vaccaro. Secret sharing schemes with veto capabilities. In *FIWAC: French-Israeli Workshop on Algebraic Coding*, 1993. 8
- [Cac95] C. Cachin. On-line secret sharing. *Lecture Notes in Computer Science*, 1995. 12
- [CCI88] CCITT. Recommendation X.500: The directory–overview of concepts, models and services, 1988. 14
- [CGL99] Richard Cleve, Daniel Gottesman, and HK Lo. How to share a quantum secret. *Physical Review Letters*, pages 1–5, 1999. 11, 12
- [CL03] Chiou and Lai. A tempo-based t-out-of-n audio cryptography scheme. *TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems*, 2003. 11
- [Com98] Federal Public Key Infrastructure Steering Committee. Access with trust. Government Information Technology Services Board, Office of Management and Budget, September 1998. 14
- [CP13] Paul Andrew Crocker and Adolfo Peixinho. Secret Sharing for High Security Codes on Smart-Cards. page 45. ECIW - European Conference on Information Warfare and Security, 2013. 3
- [CS90] Chen and Stinson. Recent results on combinatorial constructions for threshold schemes. *AJC: Australasian Journal of Combinatorics*, 1, 1990. 9

- [DCB94] Yvo Desmedt, Giovanni Di Crescenzo, and Mike Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. In *ASIACRYPT*, pages 21–32, 1994. 10
- [Des87] Yvo Desmedt. Society and Group Oriented Cryptography: A New Concept. *Advances in Cryptology — CRYPTO’ 87*, pages 120–127, 1987. 10, 11
- [Des98] Yvo Desmedt. Some Recent Research Aspects of Threshold Cryptography. *Information Security*, pages 158–173, 1998. 10
- [DF90] Yvo Desmedt and Y. Frankel. Threshold cryptosystems. In *Science*, pages 307–315. Springer, 1990. 10
- [DF94] Yvo Desmedt and Yair Frankel. Perfect homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM J. Discrete Math.*, 7(4):667–679, 1994. 10
- [DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976. 13
- [DHQ98] Yvo Desmedt, Shuang Hou, and Jean-Jacques Quisquater. Cerebral cryptography. In *Information Hiding*, pages 62–72, 1998. 11
- [ejba] [online]Available from: <http://www.ejbca.org> [cited 2013]. 23, 36
- [ejbb] [online]Available from: <http://sourceforge.net/projects/ejbca/files/WebServiceRA/> [cited 2013]. 27
- [Ell70] J.H. Ellis. The possibility of secure non-secret digital encryption. *UK Communications Electronics Security Group*, (January), 1970. 13
- [Ell99] J.H. Ellis. The history of non-secret encryption. *Cryptologia*, 1976(April):7–10, 1999. 13
- [Far10] Oriol Farras. Ideal hierarchical secret sharing schemes. *Theory of cryptography*, pages 1–24, 2010. 9
- [FD92] Y Frankel and Y Desmedt. Parallel reliable threshold multisignature. *Dept. of Elect. Eng. and Computer Sci., Univ. of . . .*, 1992. 10
- [Fel87] Paul Feldman. A Practical Scheme for Non-interactive Verifiable Secret Sharing Paul Feldman Massachusetts Institute of Technology. pages 427–437, 1987. 10
- [GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and T. Rabin. Robust Threshold DSS Signatures. *Proceedings of the 15th annual international conference on Theory and application of cryptographic techniques*, pages 1–33, 1999. 10
- [Gol05] Oded Goldreich. *Foundations of Cryptography – A Primer*. now Publishers Inc., 2005. 17
- [Got99] Daniel Gottesman. On the theory of quantum secret sharing. Technical report, Microsoft Research (MSR), 1999. 11, 12
- [GR97] David Goodman and Colin Robbins. Understanding LDAP X.500 - Version 2.0. Technical report, European Electronic Messaging Association, August 1997. 14

- [GRJK07] Rosario Gennaro, Tal Rabin, Stanislaw Jarecki, and Hugo Krawczyk. Robust and Efficient Sharing of RSA Functions. *Journal of Cryptology*, 20(3):393–393, July 2007. 10
- [HBB99] Mark Hillery, V Bužek, and A Berthiaume. Quantum secret sharing. *Physical Review A*, 1999. 11, 12
- [ISN87] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings IEEE Globecom '87*, pages 99–102. IEEE, 1987. 8, 9
- [java] [online] Available from: <http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/ScheduledExecutorService.html> [cited 2013]. 28
- [javb] [online] Available from: <http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/Executors.html> [cited 2013]. 28
- [Jul60] B. Julesz. Binocular depth perception of computer-generated patterns. *Bell System Tech.*, 39(5):1125–1161, 1960. 11
- [KGH83] Karnin, Greene, and Hellman. On secret sharing systems. *IEEE TIT: IEEE Transactions on Information Theory*, 29, 1983. 9
- [KKI99] Anders Karlsson, Masato Koashi, and Nobuyuki Imoto. Quantum entanglement for secret sharing and secret splitting. *Physics Review A*, 59:162–168, Jan 1999. 11, 12
- [Koh78] Loren M. Kohnfelder. Towards a practical public-key cryptosystem. 1978. 13
- [KOS<sup>+</sup>93] Kurosawa, Okada, Sakano, Ogata, and Tsujii. Nonperfect secret sharing schemes and matroids. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1993. 9
- [Kra93] Krawczyk. Secret sharing made short. In *CRYPTO: Proceedings of Crypto*, 1993. 12
- [Lan95] S. Langford. Threshold DSS signatures without a trusted party. *Advances in Cryptology — CRYPTO' 95*, pages 397–409, 1995. 10
- [LHB<sup>+</sup>] Arjen K Lenstra, James P Hughes, Joppe W Bos, Thorsten Kleinjung, and Christophe Wachter. Ron was wrong , Whit is right. *Observatory*, pages 1–16. 19, 21
- [Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Communications of the ACM*, 21(4):294–299, April 1978. 13
- [MM11] João Miguel and Silva Mendes. Trusted Civitas : Client Trust in CIVITAS Electronic Voting Protocol Engenharia Informática e de Computadores. 2011. 10
- [MS13] Christopher Meyer and Jörg Schwenk. Lessons learned from previous SSL/TLS attacks - A brief chronology of attacks and weaknesses. *IACR Cryptology ePrint Archive*, page 49, 2013. 61
- [mys] [online] Available from: <http://www.symantec.com/connect/articles/securing-mysql-step-step> [cited 2013]. 35
- [nis] [online] Available from: <http://www.nist.gov/> [cited 2013]. 19
- [NS94] M Naor and A Shamir. Visual cryptography. *Advances in Cryptology—EUROCRYPT'94*, 1994. 11

- [OK96] Obana and Kurosawa. Veto is impossible in secret sharing schemes. *IPL: Information Processing Letters*, 58, 1996. 8
- [Ols04] Fredrik Olsson. A lab system for secret sharing, 2004. 14
- [ope] [online]Available from: <http://www.openca.org> [cited 2013]. 23
- [OTH03] Iwamoto M. Ogawa T., Sasaki A. and Yamamoto H. Coding efficiency and construction of quantum secret sharing schemes. *Proc. of Symposium on Information Theory and Its Applications*, pages 651–654, 2003. 12
- [Per09] Colin Percival. Stronger key derivation via sequential memory-hard functions. *BSDCan 2009*, pages 1–16, 2009. 21, 22
- [qua] [online]Available from: <https://www.ssllabs.com/ssltest/> [cited 2013]. 54
- [Rab98] Tal Rabin. A Simplified Approach to Threshold and Proactive RSA. pages 89–104, 1998. 10
- [Sch] Bruce Schneier. Attack trees - modeling security threats. Available from: <https://www.schneier.com/paper-attacktrees-ddj-ft.html> [cited 2013]. x, 61, 65
- [SF13] Pratik Guha Sarkar and Shawn Fitzgerald. Attacks on SSL - A comprehensive study of BEAST , CRIME , TIME , BREACH , LUCKY 13 RC4 Biases, 2013. 61
- [Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979. 5, 6, 9, 10, 11
- [Sim93] G. J. Simmons. *Cryptology*. Encyclopaedia Britannica, fifteenth edition edition, 1993. 13
- [Sta11] William Stallings. *Cryptography and Network Security - Principles and Practice*. Prentice Hall, 5th edition, 2011. ix, 19
- [Uni97] International Telecommunication Union. ITU-T: X.509 : Information technology - Open Systems Interconnection Open Systems Interconnection – The directory: Authentication framework, June 1997. 14
- [Vat11] Nicuor Vatra. A pki architecture using opensource software for e-government services in romania. *Engg Journals Publications*, 2011. 23
- [Wah97] M. Wahl. A summary of the X.500(96) user schema for use with LDAPv3. Network Working Group - RFC-2256, December 1997. 14
- [Yam86] Hiroshige Yamamoto. Secret sharing system using  $(k, l, n)$  threshold scheme. *Electronics and Communications in Japan (Part I: Communications)*, 69(9):46–54, 1986. 9, 10
- [Yao82] Andrew C. Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE, 1982. 1
- [YDQ98] H. Shuang Y. Desmedt and J-J Quisquater. Audio and optical cryptography. *Advances in Cryptology—ASIACRYPT*, 1998. 11
- [ZLL11] Qianqian Zhang, Zhihui Li, and Xiong Li. A verifiable secret sharing scheme without dealer in vector space. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, volume 4, pages 2222–2225. IEEE, 2011. 5