Design and Implementation of Single Sign-On Based on Public-Key Infrastructure

연세대학교 대학원

전기 · 전자공학과

정 연 식

Design and Implementation of Single Sign-On Based on Public-Key Infrastructure

지도 송 홍 엽 교수

이 논문을 박사 학위논문으로 제출함

2000년 12월 일

연세대학교 대학원

전기 · 전자공학과

정 연 식

정연식의 박사 학위논문을 인준함

심사위원 _____ 인

심사위원 _____ 인

심사위원 _____ 인

심사위원 _____ 인

심사위원 _____ 인

연세대학교 대학원

2000년 12월 일

감사의 글

본 논문이 완성되기까지 세심한 지도와 격려로 이끌어주신 송홍엽 교수님께 감사 드립니다. 지금은 정년퇴임 하셨지만 많은 가르침을 주셨던 박규태 교수님께 감사 드립니다. 심사를 통하여 부족한 논문이 모습을 갖출 수 있도록 조언을 아끼지 않으신 김재희 교수님, 이재용 교수님, 김영용 교수님께 감사 드립니다. 아울러 학부 시절부터 끊임없는 지도로 학문의 길을 인도해 주셨던 김봉열 교수님을 비롯한 모든 교수님들께 존경과 감사의 말씀을 드립니다.

많은 시간을 함께 했던 디지털 연구실 선배님들과는 디지털 포럼이라는 자리를 통해 다시 뵐 수 있게 됨을 영광으로 생각합니다. 특히 졸업논문과 진로문제 등에 있어서 많은 조언과 격려를 주셨던 김인겸 선배님께 감사의 말씀을 드립니다. 3 년간의 강사생활을 뒷받침해 주셨던 강현철 선배님, 졸업을 항상 걱정해 주셨던 정현민, 김학수, 김용규, 이병래 선배님께도 감사 드립니다. 도움도 드리지 못하고 먼저 떠나게 되어 김한주 선배님께는 죄송한 마음 뿐입니다. 따뜻한 보살핌으로 연구실 후배들을 이끌어주신 윤택현, 이수종 선배님께도 감사 드리며 빠른 시일 내에 행복한 가정을 꾸미시기를 바랍니다. 대학원 생활동안 많은 도움을 주셨던 친구 같은 사수 이임건 선배님과 많은 관심과 격려를 주신 이승현 선배님께도 감사 드립니다. 얼마 전에 결혼한 이재철 선배님과 곧 결혼하게 될 서종열 선배님의 행복을 기원합니다. 먼저 졸업했지만 많은 도움을 주고 있는 부사수 이원열 학형에게도 감사 드립니다. 같은 시기에 논문을 쓰며 서로에게 격려를 아끼지 않았던 박춘석 선배님께도 진심으로 감사 드리며, 문장욱, 염창열 학형도 졸업 후의 모든 진로가 형통하기를 바랍니다. 전공을 바꾸는 데 있어 많은 도움을 준 김정헌 학형과 연구실을 이끌어가고 있는 신민호, 은유창 선배님께도 감사 드립니다. 많은 도움도 주지 못하고 축구만 하자고 괴롭혔던 홍윤표, 박성은, 최기훈, 변우성 학형과 윤현철 선배님에게도 고마움을 표하며, 황재훈 학형과 정진우 학형은 암호학 분야를 열심히 해서 선배보다 훨씬 훌륭한

논문을 쓰기를 바랍니다. 졸업논문 작업에 많은 도움을 주었던 동기 익순이와, 힘들 수도 있었던 대학원 생활동안 서로에게 의지가 되었던 현구, 용훈이, 그 외 B 모임 친구들과 91 학번 동기들에게도 고마움과 우정을 전합니다.

연세대학교에서 보낸 10 년의 세월을 돌이켜 볼 때 많은 분들로부터 분에 넘치는 사랑과 지도를 받은 것 같습니다. 이제는 학교에서 배우고 얻은 많은 것들을 사회에 나가서 많은 사람들에게 베풀며 살아가도록 노력하겠습니다.

학업에 전념할 수 있도록 믿고 보살펴 주신 장인, 장모님께 깊이 감사 드립니다. 큰 처남 내외분과 작은 처남에게도 감사 드리며, 정민이가 건강하게 자라기를 바랍니다. 동생의 학업을 위해 항상 돌봐주신 매형 내외분과 형님 내외분께도 감사 드리며, 예림이와 형섭이, 그리고 앞으로 태어날 조카도 건강하게 자라기를 바랍니다. 학생 남편을 믿고 뒷바라지해준 사랑스러운 아내에게도 깊은 사랑과 감사를 드리며, 앞으로 자상한 남편이 될 것을 약속하겠습니다. 무엇보다도 오늘의 제가 있을 수 있도록 헌신적인 사랑으로 보살펴 주신 아버님, 어머님께 진심으로 감사 드리며, 항상 건강하시기를 기도 드립니다. 이 모든 것 위에 함께 하신 하나님께 감사 드리며, 저를 사랑해주신 모든 분들께 이 조그마한 결실을 드립니다.

2000 년 12 월

정연식 드림

Design and Implementation of Single Sign-On Based on Public-Key Infrastructure

by

Yeonsik Jeong

Submitted in Partial Fulfillment of the
Requirements for the Degree
Doctor of Philosophy

Supervised by
Professor Hong-Yeop Song, Ph.D.

Department of Electrical and Electronic Engineering
The Graduate School

YONSEI University
Seoul, KOREA

2000

Contents

List of Figures iv

List of Tables vii

List of Abbreviations viii

Abstract xi

1 Introduction 1

1.1 Motivation 1

1.2 Dissertation Overview 5

2 Preliminaries 7

2.1 Cryptosystem 7

2.1.1 Secret-Key Cryptosystem 7

2.1.2 Public-Key Cryptosystem 10

2.1.3 Digital Signature 13

2.2 Cryptographic Authentication System 15

3 Design of Public-Key Infrastructure 18

3.1 Public-Key Infrastructure Overview 18

3.2 Background 20
3.2.1 X.509 Authentication Service 20
3.2.2 Lightweight Directory Access Protocol 21
3.2.3 Public-Key Cryptography Standards 22
3.3 Design of Public-Key Infrastructure 25
3.3.1 PKI Data Structures 25
3.3.2 PKI Certification Path Topology 27
3.3.3 The System Design 30

4 Design of Single Sign-On **33**
4.1 Single Sign-On Overview 33
4.2 Previous Works 34
4.2.1 Kerberos System 35
4.2.2 Netscape Single Sign-On 37
4.3 The Proposed Single Sign-On 39
4.3.1 The System Overview 39
4.3.2 Technique for Performing an Authentication Protocol 41
4.3.3 Requirements Capture of the Proposed System 42

5 Implementation of Single Sign-On **57**
5.1 Total System Overview 57
5.2 Implementation of Public-Key Infrastructure 58
5.2.1 Implementation of CA Server 58
5.2.2 Implementation of LDAP Server 62
5.2.3 Implementation of Client Module 64

5.3 Implementation of Single Sign-On 67

5.3.1 Implementation of Application Server 68

5.3.2 Implementation of Authentication Server 70

5.3.3 Implementation of Client 73

5.3.4 Verification and Validation of the Proposed System 74

6 Conclusion **75**

References **77**

A System Description in SDL **84**

Summary (in Korean) **97**

List of Figures

2.1 Cipher block chaining(CBC) mode. 9

2.2 Public-key Cryptosystem. 12

2.3 Two approaches to digital signatures: || means concatenation. 14

2.4 Challenge-response protocol with RSA algorithm. 16

3.1 X.500 information model. 22

3.2 X.509 Formats. 26

3.3 PKI certification path topology: $\longrightarrow$ means certification, and $\longleftrightarrow$ means cross-certification. 28

3.4 PKI system. 30

4.1 Overview of the Kerberos system. 37

4.2 Netscape's SSO system. 38

4.3 The proposed SSO system. 40

4.4 Scenario 1: General case. 44

4.5 Scenario 2: In case a user enters invalid password. 46

4.6 Scenario 3: In case the LDAP directory lookup fails. 47

4.7 Scenario 4: In case the signature has been modified on transmission. . . 48

4.8 Scenario 5: In case a user accesses the same application server from the same client within timer expiration. 49

4.9 Scenario 6: In case a user accesses the same application server from the same client after the timer expires. 51
4.10 Scenario 7: In case a user accesses another application server from the same client within timer expiration. 52
4.11 Scenario 8: In case a user accesses another application server from the same client after the timer expires. 53
4.12 Scenario 9: In case an opponent accesses the application server from the another client within timer expiration. 54
4.13 Scenario 10: In case an opponent directly accesses an application server without authentication. 55
4.14 Scenario 11: In case a user sends multiple service_req messages. 56

5.1 The proposed and implemented single sign-on system. 58
5.2 The initial page of the CA server. 60
5.3 The proposed LDAP structure. 63
5.4 DES key and IV generation using password-based encryption algorithm. 66
5.5 State transition diagram of an application server. 69
5.6 State transition diagram of an authentication server. 71

A.1 Organizer View of the proposed SSO system. 84
A.2 System diagram of the proposed SSO system. 85
A.3 Block diagram of the application server. 86
A.4 Process diagram of the application server. 87
A.5 Block diagram of the authentication server. 88
A.6 Process diagram of the authentication server. 89
A.7 Process diagram of the authentication server: *continued.* 90

A.8 Process diagram of the authentication server: *continued.* 91

A.9 Block diagram of the client. 92

A.10 Process diagram of the client. 93

A.11 Process diagram of the client: *continued.* 94

A.12 Block diagram of the LDAP server. 95

A.13 Process diagram of the LDAP server. 96

List of Tables

2.1 Applications for public-key cryptosystems. 13

4.1 Messages exchanged between system components: – means that the message has no parameters. 43

5.1 Implementation environment. 59

5.2 The values used in the X.509 certificate generated by the proposed CA server. 60

List of Abbreviations

AES Advanced Encryption Standard

ASCII American Standard Code for Information Interchange

CA Certificate Authority

CBC Cipher Block Chaining

CFB Cipher FeedBack

CRL Certificate Revocation List

CGI Common Gateway Interface

DAP Directory Access Protocol

DES Data Encryption Standard

DN Distinguished Name

DSS Digital Signature Standard

ECB Electronic CodeBook

FIPS Federal Information Processing Standard

HTTP HyperText Transfer Protocol

IMAP Internet Mail Access Protocol

IPSEC IP SECurity

ISO International Organization for Standardization

ITU International Telecommunications Union

IV Initial Vector

LDAP Lightweight Directory Access Protocol

MAC Message Authentication Code

MSC Message Sequence Chart

MSL Maximum Segment Lifetime

MIME Multipurpose Internet Mail Extension

NIST National Institute of Standards and Technology

NNTP Network News Transfer Protocol

OFB Output FeedBack

OSI Open Systems Interconnection

PIN Personal Identification Number

PKCS Public-Key Cryptography Standards

PKI Public-Key Infrastructure

RSA Rivest, Shamir, and Adleman

SDL Specification and Description Language

SET Secure Electronic Transaction

SGT Service-Granting Ticket

SLAPD Stand-alone LDAP Daemon

SSL Secure Socket Layer

SSO Single Sign-On

STD State Transition Diagram

TGS Ticket-Granting Server

TGT Ticket-Granting Ticket

TLS Transport Layer Security

WWW World Wide Web

Abstract

Design and Implementation of Single Sign-On Based on Public-Key Infrastructure

Yeonsik Jeong

Dept. of Electrical and Electronic Eng.

The Graduate School

Yonsei University

With the explosive growth of the Internet, authentication scheme is becoming more and more important issue in the security area. Cryptographic solution to the problems of both password-based authentication and multi-server authentication has been researched, which leads to the birth of a single sign-on system.

In this dissertation, a single sign-on system has been designed and implemented. The purpose of the proposed system is that a user logs in once using a single password, and gets authenticated access to all the servers that he is authorized to use, without sending any passwords over the network. The proposed single sign-on system involves the use of digital signature and cryptographic authentication protocols. For its implementation, a public-key infrastructure is also designed and implemented.

The proposed single sign-on system consists of an authentication server, multiple

application servers and clients, and public-key infrastructure which is composed of a certificate authority server, a directory server, and client modules. To view the behavior of the proposed system, message sequence charts and state transition diagrams have been presented.

Implementation results show that the proposed single sign-on system can be easily implemented on the Intranet, in which most services are provided through world wide web servers and browsers and require strong authentication, only by taking the client module into the browsers and it works well with a little delay during log-in.

Key Words: authentication, single sign-on, digital signature, public-key infrastructure

Chapter 1

Introduction

1.1 Motivation

Modern computer systems require the ability to accurately identify the user requesting a service in order to protect the contents of the service from opponents. In traditional systems, the user's identity is verified by checking a password entered during log-in. Then, the system records the identity and uses it to determine what operations may be performed. This process of verifying the user's identity is called authentication. Password-based authentication, which has been widely used in most systems, is not suitable for use on current networks, in which electronic commerce prospers on large scale. Passwords sent over the network may be intercepted and subsequently used by eavesdroppers to impersonate the user.

Many of the weakness of password-based authentication are due to the fact that passwords can be reused again. One solution to the problems is to make passwords expire as soon as they are used. The user is given a list of passwords, and when a password is used it is removed from the list and the next password on the list is chosen as the current password. The most important point here is to make sure that it is impossible or

infeasible to deduce the following password if one has access to the previous password that was used. Correctly deployed, one-time password scheme can be effective, but it has some drawbacks: It requires increased administration and secure storage [1].

Cryptographic techniques have also been introduced to strengthen authentication scheme, which are called challenge-response protocol [2, 3]. The main idea is that the prover proves his identity to the verifier without revealing the secret information during the protocol. Generally, challenge-response protocols work as follows: The verifier sends a time-variant challenge to the prover. The prover computes a response from the challenge using some encryption algorithm, which is secret-key or public-key algorithm, and sends the response to the verifier. So, the response depends on the challenge and the secret information.

With the very rapid growth of the Internet and Intranet, a variety of services from multiple servers have been provided. Users are required to use a different password to be authenticated to each server they are willing to access. Separate password per each server is an ongoing problem to both users and system administrators. Even though cryptographic authentication schemes are used, it still remains a burden to implement those into each server. In addition, a user should also perform the authentication process with each server that he is to access, without respect to the type of cryptosystems used.

Due to these practical shortcomings, a concept of Single Sign-On(SSO) has been introduced, which tries to integrate authentication schemes, which each server implemented respectively, inside domain of services. Implementation of SSO requires some way for a user to log in once using a single password, and get authenticated access to all the servers that he is authorized to use, without sending any passwords over the network. The idea behind SSO is that the identity of a user is checked only once and the

electronic identity is transferred to different computing resources securely and automatically. The rights associated with the users and resources are administered in a central manner, so there is no need for the system administrators to worry about and manage several databases [4, 5].

Various systems have recently been proposed to support the SSO, but two systems are widely used for the implementation: One is Kerberos system based on secret-key cryptography, the other is Netscape SuiteSpot server using public-key cryptography.

Kerberos was developed in the mid-'80s as part of MIT's Project Athena [6] and is one of the most widely used authentication services. Kerberos provides a centralized authentication server whose function is to authenticate users to servers and vice versa, using a secret-key cryptosystem. It provides secure network authentication for TCP/IP networks, allowing a user to access different servers on the network [7].

However, limitations of Kerberos have been described in the literature [8]. It may be possible to cache and replay old authenticators, called ticket. In addition, because it is based on the secret-key cryptography, it has troubles with key management. Work has also been underway to enhance the Kerberos authentication, using the public-key cryptography and smart-card interface for key management [9].

The benefits of public-key cryptography are that it provides for encryption to keep information confidential, and authentication, data integrity, and nonrepudiation through digital signatures. Therefore, public-key cryptography is uniquely qualified to satisfy the necessary security requirements for the authentication services, and it has become the preferred means for providing these capabilities.

As more applications use public-key cryptography, it leads to a requirement for a Public-Key Infrastructure(PKI), which is a comprehensive system that provides the

public-key-based encryption and digital signature services on behalf of the applications [10, 11]. A major benefit of the PKI is that it enables the use of encryption and digital signature services in a consistent manner across a wide variety of applications. A PKI is also a certificate management infrastructure which issues and provides access to public-key certificates.

Netscape has developed total authentication system, called SuiteSpot server, for supporting its products such as Netscape Communicators and Netscape Enterprise servers. Netscape's approach [12] to SSO involves the use of digital certificates for the authentication of users to servers. For Netscape products, the SSO provides an authentication mechanism that replaces more cumbersome multiple-password authentication without affecting existing access-control mechanisms.

Netscape's authentication scheme is based on the use of Secure Socket Layer(SSL); It uses the handshake protocol of SSL, which allows the server and client to authenticate each other and to negotiate an encryption and Message Authentication Code(MAC) algorithm and cryptographic keys to be used to protect data sent in an SSL payload. For the implementation, it replaces client authentication of the handshake protocol based on the machine's certificate with that based on the user's certificate.

In this dissertation, a SSO system based on both PKI and cryptographic authentication protocol has been designed and implemented. The proposed system seems to be analogous to the Netscape's scheme, in that both systems aim at the World Wide Web(WWW) services and are based on the public-key certificates. But Netscape's approach is rather analogous to that of Kerberos, except for using the public-key cryptography. In both systems, for a user to access the server from which he wants to get a service, he must access a central server, which is the authentication server in the Ker-

beros or the SuiteSpot server in the Netscape's SSO, and obtain the evidence that he is authorized to use the servers inside the domain, such as Intranet.

In the proposed SSO, we employ the mechanism that a user directly connects the server he wants to access and then the server automatically requests an authentication service from the central server, which is the authentication server, using the location header of HTTP protocol. In addition, the proposed SSO system is implemented entirely on the application layer, so independent of the lower layer implementations, unlike the Netscape's SSO which is based on the transport layer, such as SSL. Therefore, it is possible to implement an application-specific authentication scheme.

As standard web browsers, such as Internet Explorer or Netscape Communicator, do not have the capabilities to interact with the PKI, extra functionality for communicating with the PKI and the proposed authentication services should be additionally implemented. In the proposed implementation, we employ a Microsoft ActiveX control programming technique for the client module.

Implementation results show that the proposed SSO can be easily implemented on the Intranet, in which most services are provided through WWW servers and browsers, only by taking the client module into the browsers and it works well with a little delay during log-in.

1.2 Dissertation Overview

The outline of this dissertation is as follows: We begin in Chapter 2 with a brief discussion of the preliminary knowledge on the cryptosystem and authentication system. In Chapter 3, we describe a PKI in general, and design a PKI which will be employed in

the proposed SSO system.

We begin with Chapter 4 by discussing two previous works on the SSO: Kerberos by MIT and Netscape's SuiteSpot SSO. Then, we design a SSO based on both the PKI and cryptographic authentication protocol. The requirements for designing the proposed SSO are also provided in more detail. In Chapter 5, we implement two core systems: PKI and SSO. Details about implementing the systems are also described. Finally, Chapter 6 ends with the conclusion and further study.

Chapter 2

Preliminaries

This chapter provides some background knowledge for understanding the proposed SSO system in this dissertation. We describe the general properties of cryptosystem and cryptographic authentication system in this chapter. The specific knowledge related to the implementation of the PKI and SSO will be mentioned in the next chapters.

2.1 Cryptosystem

2.1.1 Secret-Key Cryptosystem

Secret-key cryptosystem is the classical form of cryptography that has been used since ancient times. For secret-key cryptosystem to work, the two parties to exchange information must share a secret: the key used for encryption and decryption. This requires prior communication between two parties over a secure channel in order to agree on the key. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised.

There have been many secret-key cryptosystems, but most of them in current use are based on a structure referred to as a Feistel block cipher, which uses the concept of

substitution and permutation.

Feistel [13] proposed the use of a cipher that alternates substitutions and permutations. In fact, this is a practical application of a proposal by Shannon [14] to develop a product cipher that alternates confusion and diffusion functions.

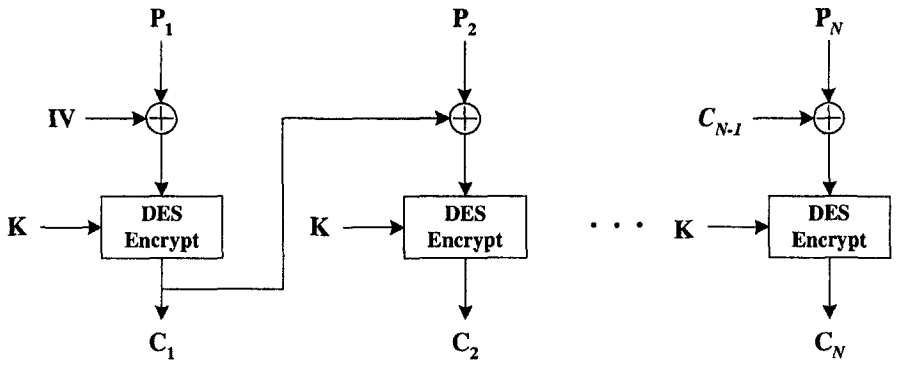
The most widely used encryption scheme is the Data Encryption Standard(DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology(NIST), as Federal Information Processing Standard 46(FIPS PUB 46) [15].

For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

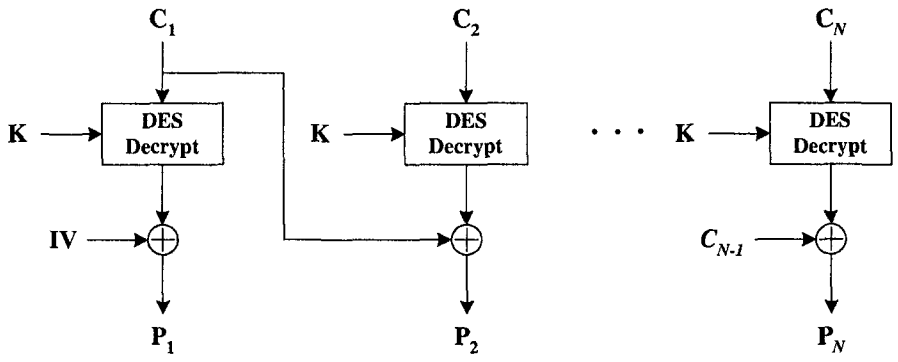
The DES algorithm is a basic building block for providing data security. To apply DES in a variety of applications, four modes of operation have been defined in FIPS PUB 74, 81 [16, 17]: Electronic CodeBook(ECB) mode, Cipher Block Chaining(CBC) mode, Cipher FeedBack(CFB) mode, and Output FeedBack(OFB) mode.

Among four modes, we focus on the CBC mode, which is used for general block-oriented transmission and authentication [18, 19]. In this mode, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same key is used for each block. Note that the input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block. Therefore, repeating pattern of 64 bits are not exposed. The block diagram of CBC mode is shown in Figure 2.1.

With recent successful key exhaustive attacks, the useful lifetime of DES is now drawing to a close [20, 21]. In 1996, NIST began preparing for development of a suc-



(a) Encryption



(b) Decryption

Figure 2.1: Cipher block chaining(CBC) mode.

cessor standard, and announced the initiation of a process to develop the Advanced Encryption Standard(AES) in 1997. Various algorithms has been proposed and competed for inclusion in the AES. Most candidates are designed and modified to support 128-bit blocks of plaintext to enhance the existing algorithms for providing a replacement for DES, which were limited to having a 64-bit block size.

2.1.2 Public-Key Cryptosystem

In contrast with secret-key cryptography, public-key cryptography is comparably new. The concept of public-key cryptography was invented by Diffie and Hellman [22]. Their contribution was the notion that keys could come in pairs–an encryption key and a decryption key–and that it could be infeasible to generate one key from the other. Rivest, Shamir, and Adleman invented the first implementation of the public-key cryptography in 1977 and published in 1978 [23]. The RSA scheme has been accepted and implemented as the only general-purpose approach since that time. The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and $n - 1$ for system parameter n.

Public-key algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic:

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and encryption key.

Some algorithms, such as RSA, also have the additional characteristic:

- Either of two related keys can be used for encryption, with the other for decryption.

The use of public-key cryptosystems can be classified into three categories:

- Encryption/Decryption: The sender encrypts a message with the recipient's public-key.
- Authentication: The sender signs a message with its private-key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.

- Key exchange: Two parties cooperate to exchange a session key.

There is some source A for a message X, which is intended for destination B. B generates a related pair of keys: a public-key, $\mathrm{KU_b}$, and a private-key, $\mathrm{KR_b}$. $\mathrm{KR_b}$ is known only to B, whereas $\mathrm{KU_b}$ is publicly available and therefore accessible by A. The process for encryption and decryption can be written as

$$Y = E_{\mathrm{KU_b}}(X)$$

$$X = D_{\mathrm{KR_b}}(Y),$$

where $E(\cdot)$ is an encryption algorithm and $D(\cdot)$ is a decryption algorithm. The process for encryption and decryption is demonstrated in Figure 2.2: (a).

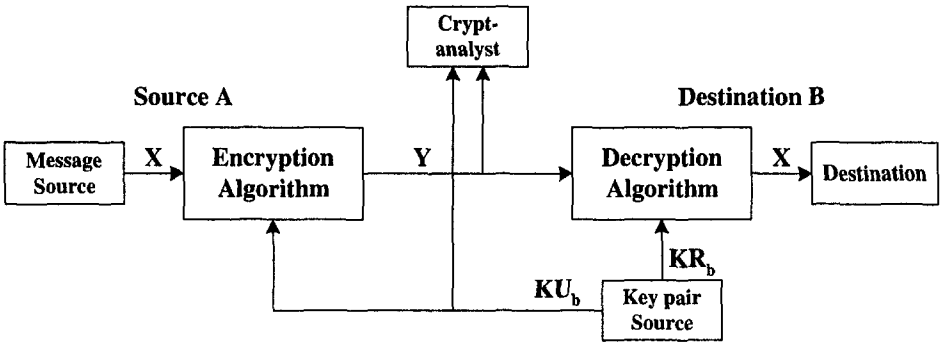
Figure 2.2: (b) shows the use of public-key cryptosystem to provide authentication:

$$Y = E_{\mathrm{KR_a}}(X)$$

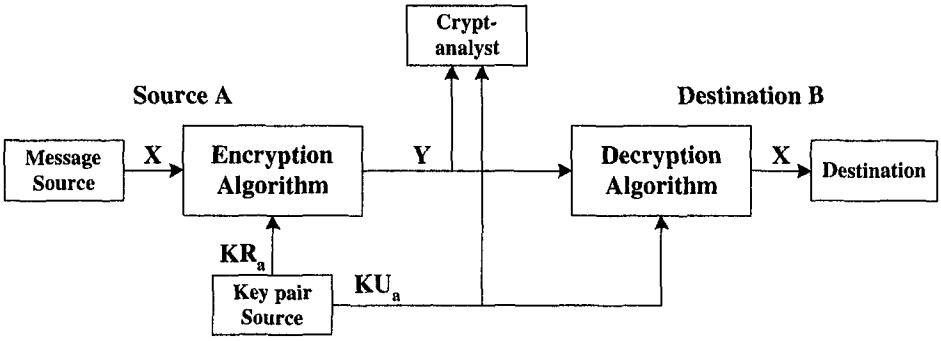
$$X = D_{\mathrm{KU_a}}(Y).$$

In this case, A prepares a message to B and encrypts it using A's private-key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private-key, only A could have prepared the message. Therefore, the entire encrypted message serves as a digital signature. In addition, it is impossible to alter the message without access to A's private-key, so the message is authenticated in terms of data integrity.

One of the important applications of public-key cryptosystem is the key exchange. The first published public-key algorithm by Diffie and Hellman that defined the concept of public-key cryptography is generally referred to as Diffie-Hellman key exchange [22, 24]. The purpose of the algorithm is to enable two users to exchange a key securely that



(a) Encryption/Decryption



(b) Authentication

Figure 2.2: Public-key Cryptosystem.

can then be used for subsequent encryption of messages using secret-key cryptography. A number of commercial products employ this key exchange technique.

Some algorithms are suitable for all three applications, whereas others can be used only for one or two of these applications. Table 2.1 indicates the applications supported by the algorithms discussed in this chapter.

In practice, computing a public-key cipher takes much longer than a secret-key cipher; usually that is too slow to support bulk data encryption. So hybrid cryptosystems are introduced: A secret-key cryptography with a random session key is used to encrypt

Algorithm	Encryption/Decryption	Digital signature	Key exchange
RSA	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Table 2.1: Applications for public-key cryptosystems.

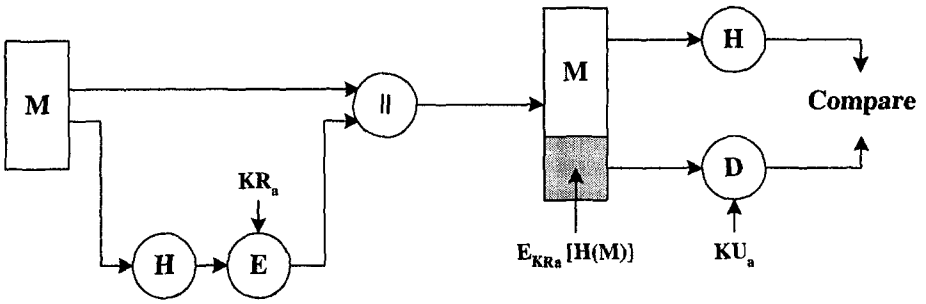
the message, and a public-key cryptography is used to encrypt the random session key.

2.1.3 Digital Signature

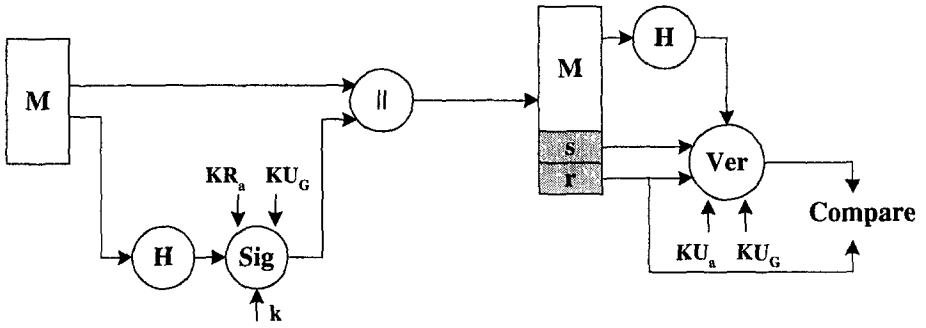
The very nature of public-key cryptography permits a form of message signing. A digital signature scheme allows one to sign the message with his private-key in such a way that everyone can verify the validity for the signature using his public-key, but no one else can forge signatures of another message. Digital signatures were originally defined by Diffie and Hellman [22], and the first implementation was based on the RSA trapdoor function [23].

The NIST has published FIPS PUB 186 [25], known as the Digital Signature S-tandard(DSS). The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange.

Figure 2.3 contrasts the DSS approach for generating digital signatures to that used with RSA. In the RSA approach, the message(M) to be signed is input to a hash function(H) that produces a secure hash code of fixed length. This hash code is then encrypted using the signer's private-key to form the signature. Both the message and the signature are then transmitted. The verifier takes the message and produces a hash code. The verifier also decrypts the signature using signer's public-key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only



(a) RSA approach



(b) DSS approach

Figure 2.3: Two approaches to digital signatures: || means concatenation.

the signer knows the private-key, only the signer could have produced a valid signature.

The DSS is based on the difficulty of computing discrete logarithms and on schemes originally presented by ElGamal [26, 27]. The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number k which is generated for this signature. The signature function(Sig) also depends on the signer's private-key and a set of parameters known to a group of communicating principals that are denoted by global public-key(KU_G). The result is a signature consisting of two components, labeled s and r.

At the receiving end, the hash code of the incoming message is generated. This

plus the signature is input to a verification function(Ver). The verification function also depends on the global public-key as well as the signer's public-key. The output of the verification function is a value that is equal to the signature component r, if the signature is valid. The signature function is such that only the signer, with the knowledge of the private-key, could have produced the valid signature.

2.2 Cryptographic Authentication System

Client authentication is an essential element of network security within most Intranets or Extranets. Almost existing servers employ an ID and password mechanism as a client authentication. In this form of authentication, password the user enters may be sent over the network in the plaintext. Even though network security protocols, e.g., SSL, IP Security(IPSEC), are applied, the passive capture of an encrypted password and its subsequent retransmission may be possible to get an authenticated access.

To prevent eavesdropping a plaintext password as well as replay attack for an encrypted password, cryptographic protocols using random number have been introduced. This *cryptographic protocol* is called a *challenge-response protocol*.

The main idea behind challenge-response protocols is that the prover proves his identity to the verifier without revealing the secret information during the protocol; the prover demonstrates to the verifier that he has the knowledge of the secret information. The challenge-response protocols used for client authentication can be divided into the following categories on the basis of their fundamental cryptographic algorithms [3]:

- Challenge-response protocols with secret-key cryptography, e.g., DES;

  In this scheme, it is assumed that a secret information is present as a key in both

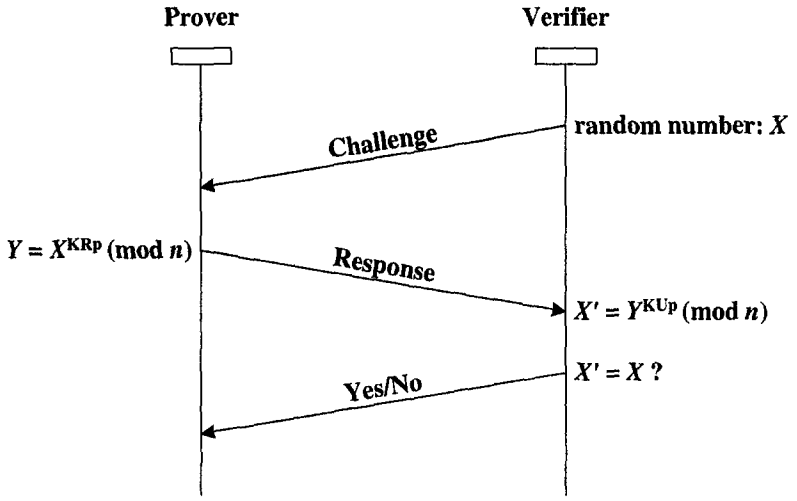


Figure 2.4: Challenge-response protocol with RSA algorithm.

the prover and the verifier. The prover proves its identity by encrypting a random number obtained from the verifier with the secret-key and sending the encrypted random number back to the verifier. Because the verifier also knows the secret-key, it can verify whether the encryption was performed with the right key and therefore, whether the user is authentic.

- Challenge-response protocols with public-key cryptography, e.g., RSA;

  In this scheme, it is assumed that the prover has a pair of key. One key of the pair represents the prover's secret information(KR_p). The other key(KU_p) is public and known, in particular, to the verifier for verifying the prover's identity. The prover uses the private-key to sign a random number received from the verifier and sends the signed random number back to the verifier. If the verifier is capable of decrypting the previously transmitted signature successfully, it is certain that only the genuine prover can possibly have carried out the signing. This protocol is based on the prover's digital signature from a random number obtained from

the verifier and shown in Figure 2.4.

- Zero-knowledge-based protocols, e.g., Fiat-Shamir algorithm [28].

  More recently, public-key-based authentication methods have been developed using models from zero-knowledge theory. In these cases, tools taken from complexity and probability theories are used to demonstrate how an interactive challenge-response protocol can be developed so that no knowledge about the secret information is exposed, no matter how many times the prover is challenged by the verifier. A number of practical methods have now been introduced more or less based on zero-knowledge technique [29, 30, 31]. Besides the methods requiring a public-key for verification of a prover, alternatives are also possible where the verification key can be deduced from the identification word of the prover; this is called identity-based authentication algorithm [32].

Chapter 3

Design of Public-Key Infrastructure

3.1 Public-Key Infrastructure Overview

Digital signatures allow us to authenticate ourselves as well as provide integrity of the data that we sign. In addition, public-key cryptography can be used to exchange cryptographic keys that are used to encrypt data, thus ensuring their confidentiality. For the system to work, we need to be certain that we have the right public-key for the person we are to communicate with. Public-key certificates allow us to bind an identity to a public-key using a trusted third party called Certificate Authority(CA). Certificates generated by a CA have the following characteristics:

- Any user with access to the public-key of the CA can recover the user's public-key that was certified;
- No party other than the CA can modify the certificate without being detected.

Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

Each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for example, in case the user's private-key is assumed to be compromised, or the user is no more certified by this CA, or the CA's certificate is assumed to be compromised. Each CA maintains a list consisting of all revoked but not expired certificates issued by that CA. These Certificate Revocation List(CRL)s should also be posted in the directory.

As more applications use public-key cryptography, it leads to a requirement for a PKI, which is a comprehensive system that provides the public-key-based encryption and digital signature services on behalf of the applications. A major benefit of the PKI is that it enables the use of encryption and digital signature services in a consistent manner across a wide variety of applications. A PKI is also a certificate management infrastructure which issues and provides access to public-key certificates. A PKI can enable its users to:

- verify the identity of a message sender based on the sender's certificate;
- verify message integrity and detect message alteration;
- exchange keys for confidentiality.

PKI is generally composed of the following components: CA servers which generate, issue, and manage certificates and CRLs, directory servers which maintain certificates, CRLs and the user information , and PKI client modules which enable users to use the PKI environments [33].

If all users subscribe to the same CA, then there is a common trust of that CA. But if there is a large community of users, it may not be practical for all users to subscribe

to the same CA. With many users, it may be more practical for there to be a number of CAs, each of which securely provides its public-key to some fraction of the users. Therefore, in a PKI, trust is transferred by certification path, that is along chains of certificates. A digital signature is verified by verifying the signatures on a chain of certificates until the certificate of a CA is reached that is trusted by the verifier, and whose public-key is known to the verifier. CAs can issue certificates to each other in a systematic and ordered way or in a more flexible and less ordered way [10, 34].

3.2 Background

This section provides an overview of key techniques which make up PKI: X.509 authentication service, lightweight directory access protocol, and public-key cryptography standards.

3.2.1 X.509 Authentication Service

X.509 [35] is the authentication framework designed to support X.500 directory services. Both X.509 and X.500 are part of the X series of international standards proposed by the ISO and ITU. The X.500 standard is designed to provide directory services on large computer networks, and X.509 provides a framework for authenticating the X.500 directory which may serve as a repository of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety context: S/MIME, IPSEC, SSL/TLS, and SET. X.509 is based on the use of public-key cryptography and digital signatures and does not state the use of a specific cryptographic algorithm as well as a specific hash

algorithm.

Three basic data structures are defined in the X.509 standard: certificate, cross-certificate, and CRL. The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted CA and placed in the directory by the CA or the user.

3.2.2 Lightweight Directory Access Protocol

X.500 [36] defines the Directory Access Protocol(DAP) used by clients to access the directory. DAP is a full OSI protocol that contains extensive functionality, much of which is not used by most applications. X.500 defines the following components:

- Information model: determines the form and character of information in the directory;
- Namespace: allows the information to be referenced and organized;
- Functional model: determines what operations can be performed on the information.

The information model is centered around entries, which are composed of attributes. Each attribute has a type and one or more values. The type determines the attribute's syntax, which defines what kind of information is allowed in the values.

Entries are arranged in a tree structure and can be divided among servers in a geographical organizational distribution. Entries are named according to their position in this hierarchy by a distinguished name(DN). Alias entries, which point to other entries, are also allowed. Figure 3.1 depicts the relationship between entries, attributes, and values and shows how entries are arranged into a tree.

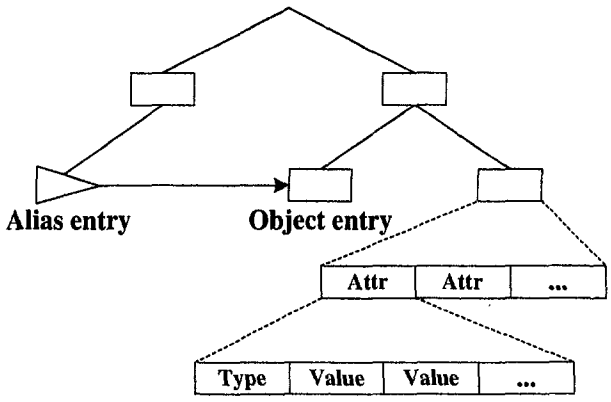


Figure 3.1: X.500 information model.

Functionally, X.500 defines operations for searching, reading, and writing directory information. Operations are also provided to add entries, delete entries, and modify existing entries.

But, the size and complexity of DAP make it difficult to run on most clients. The Lightweight Directory Access Protocol(LDAP) [37, 38] was designed to remove some of the burden of X.500 access from directory clients, making the directory available to a wider variety of machines and applications. LDAP was originally developed as a front-end to the X.500 directory. Naturally, it assumes the same information model and namespace as X.500.

LDAP has many advantages over DAP [39, 40]. First, the functional model is less complicated. Second, LDAP runs directly over TCP or other reliable transport layer. Third, LDAP uses a simplified string representation for most syntaxes.

3.2.3 Public-Key Cryptography Standards

As public-key cryptography begins to be used widely, the need to the interoperable s-tandard is increased. Even though most of vendors may agree on the basic public-key

techniques, compatibility among implementations can not be guaranteed. Interoperability requires an agreed-on standard format for encrypted and transferred data [41]. Public-Key Cryptography Standards(PKCS) cover most of the techniques related to the public-key cryptography: RSA cryptosystem, Diffie-Hellman key agreement protocol, password-based encryption, extended-certificate syntax, cryptographic message syntax, private-key information syntax, and certification request syntax. In this section, we will mention the standards directly related to the design and implementation of a PKI.

PKCS #1: RSA Encryption Standard

PKCS #1 [42] describes a method for encrypting data using the RSA public-key cryptosystem. Its intended use is in the construction of digital signatures and digital envelopes, as will be used in PKCS #7.

- For digital signatures, the content to be signed is first reduced to a message digest with a message-digest algorithm such as MD5 [43], and then an octet string containing the message digest is encrypted with the RSA private-key of the signer. The content and the encrypted message digest are represented together according to the syntax in PKCS #7 to generate a digital signature.

- For digital envelopes, the content to be enveloped is first encrypted under a content-encryption key with a content-encryption algorithm such as DES, and then the content-encryption key is encrypted with the RSA public-key of recipient. The encrypted content and the encrypted content-encryption key are represented together according to the syntax in PKCS #7 to generate a digital envelope.

PKCS #1 also describes a syntax for RSA public-keys and private-keys. The public-key syntax is used in X.509 certificates. The private-key syntax is used typically in

yielding encrypted private-keys in PKCS #8.

PKCS #5: Password-Based Encryption Standard

PKCS #5 [44] describes a method for encrypting an octet string with a secret-key derived from a password. The result of the method is also an octet string. Although PKCS #5 can be used to encrypt arbitrary octet string, its intended primary application to public-key cryptography is for encrypting private-keys when transferring them between computer systems or saving them into computer systems.Encryption algorithms employ DES encryption in CBC mode, where the secret-key is derived from a password with the MD4 [45] or MD5 message-digest algorithm.

PKCS #7: Cryptographic Message Syntax Standard

PKCS #7 [46] describes a general syntax for data to which cryptography may be applied, such as digital signatures and digital envelopes. The syntax admits recursion, that is one envelope can be nested inside another, or one can sign some previously enveloped data. One important application of PKCS #7 is to provide a message format for distributing signed certificates and CRL. The values produced according to PKCS #7 are intended to be BER-encoded, which means that the values are typically represented as octet string. While many systems are capable of transmitting arbitrary octet string reliably, it is well known that many electronic-mail systems are not. PKCS #7 does not address mechanisms for encoding octet string into ASCII string. RFC 1421 [47] can be a solution to this problem.

PKCS #8: Private-Key Information Syntax Standard

PKCS #8 [48] describes a syntax for private-key information. Private-key information includes a private-key for some public-key algorithm and a set of attributes. PKCS #8 also describes a syntax for encrypted private-keys. A password-based encryption algorithm, which is described in PKCS #5, can be used to encrypt the private-key encryption.

PKCS #10: Certification Request Syntax Standard

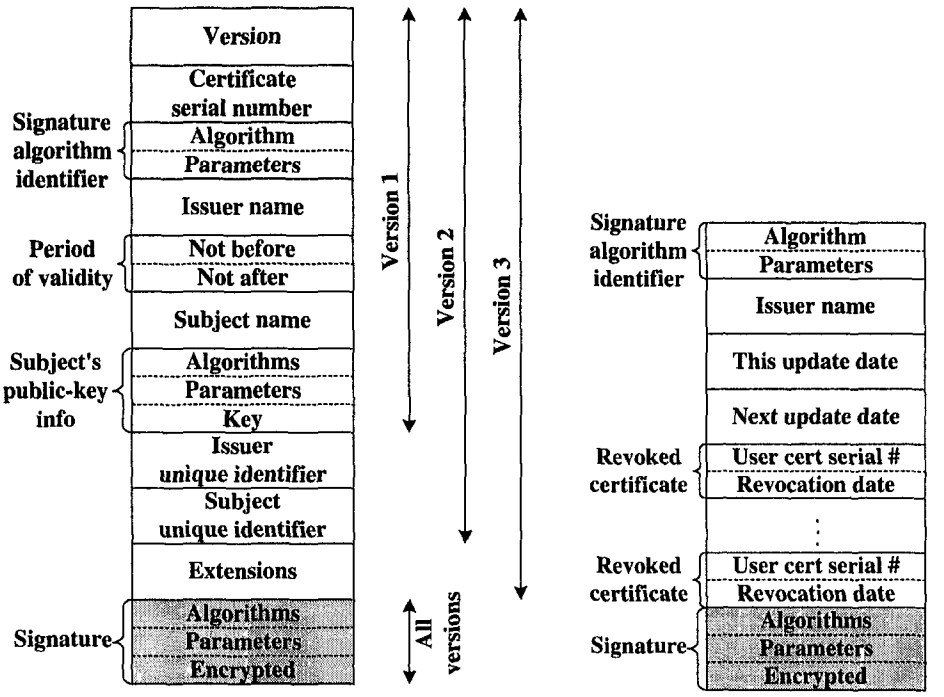
PKCS #10 [49] describes a syntax for certification requests. A certification request consists of a distinguished name, a public-key, and optionally a set of attributes, collectively signed by the entity requesting certification. Certification requests are sent to a CA, who transforms the request to an X.509 certificate.

3.3 Design of Public-Key Infrastructure

3.3.1 PKI Data Structures

Certificates, cross-certificates, and CRLs are all directory attributes from the X.509 standard. The primary purpose of an X.509 certificate is to associate a user's public-key and his name. Public-keys may be used to verify digital signatures, or to manage keys used with secret-key algorithm for confidentiality. The general format of a certificate is shown in Figure 3.2: (a) and includes the following elements:

- Version: The X.509 version that the certificate conforms to; the default is version 1;



(a) X.509 certificate

(b) Certificate revocation list

Figure 3.2: X.509 Formats.

- Certificate serial number: A unique integer value assigned to the certificate by the issuing CA;
- Signature algorithm identifier: The algorithm used to sign the certificate, together with any associated parameters;
- Issuer name: The X.500 name of the CA that created and signed this certificate;
- Period of validity: A pair of dates between the certificate is considered valid;
- Subject name: The X.500 name of the user who holds the corresponding private-key;
- Subject's public-key information: The value of the subject's public-key, plus an

identifier of the algorithm for which this key is to be used, together with any associated parameters;

- Signature: Covers all of the other fields of the certificate; it contains the hash code of the other fields, encrypted with the CA's private-key.

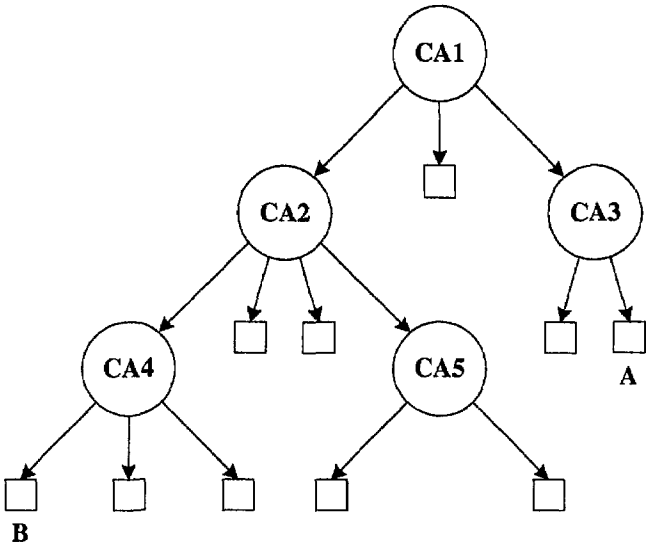
In case there exist multiple CAs, CAs may cross-certify each other, that is each CA issues the other a certificate and combine the two certificates in a single entry in the directory. The directory entry for each CA includes two types of certificates:

- Forward certificates: Certificates of CA X generated by other CAs;
- Reverse certificates: Certificates generated by CA X that are the certificates of other CAs.

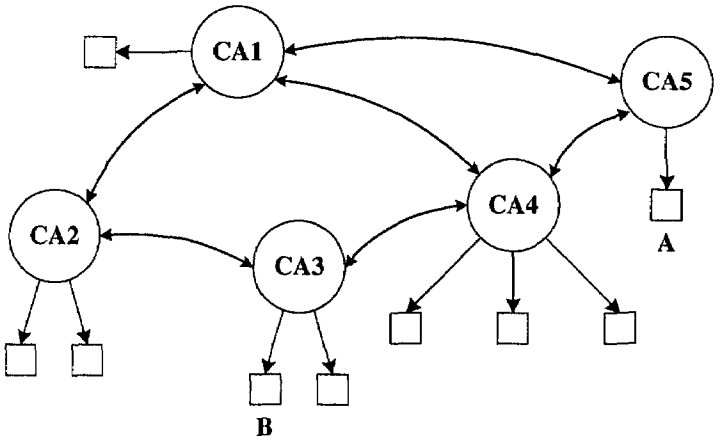
Each CRL issued and signed by the issuer includes the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. The general format of a CRL is also shown in Figure 3.2: (b).

3.3.2 PKI Certification Path Topology

The systematic, ordered topology of certification paths that is normally employed is a hierarchy, while the more general topology is a network of cross-certified CAs. The two alternatives are illustrated in Figure 3.3 and described as:



(a) Hierarchical topology



(b) Network topology

Figure 3.3: PKI certification path topology: $\longrightarrow$ means certification, and $\longleftrightarrow$ means cross-certification.

- Hierarchical topology: CAs are arranged hierarchically under a root CA that issues certificates to subordinate CAs. These CAs may issue certificates to CAs

below them in the hierarchy, or to users. In a hierarchical PKI, the public-key of the root CA is known to every user, and any user's certificate may be verified by verifying the certification path of certificates that leads back to the root CA. In Figure 3.3: (a), A verifies B's certificate, issued by CA4, then CA4's certificate, issued by CA2, and then CA2's certificate issued by CA1, the root CA, whose public-key he has known;

- Network topology: Independent CAs issue certificates each other, resulting in a general network of trust between CAs. Figure 3.3: (b) illustrates a network of CAs. A user knows the public-key of a CA near himself, generally the one that issued his certificate, and verifies the certificates of other users by verifying a certification path of certificates that leads back to that trusted CA. For example, A knows the public-key of CA5, while B knows that of CA3. There are several certification paths that lead from B to A, but the shortest path requires A to verify B's certificate, issued by CA3, then CA3's certificate issued by CA4 and finally CA4's certificate, issued by CA5. CA5 is A's CA, and he trusts CA5 and has known its public-key.

Each topology has some advantages and disadvantages over the other topology. It seems that a hierarchical topology is more appropriate, as it provides for clear lines of CAs and ability to efficiently navigate the structure for certification paths. But the PKI should be designed to minimize the length of the certification path to be verified. Therefore, the PKI mechanism which combines both the hierarchical and the network topologies has been generally used [10, 34].

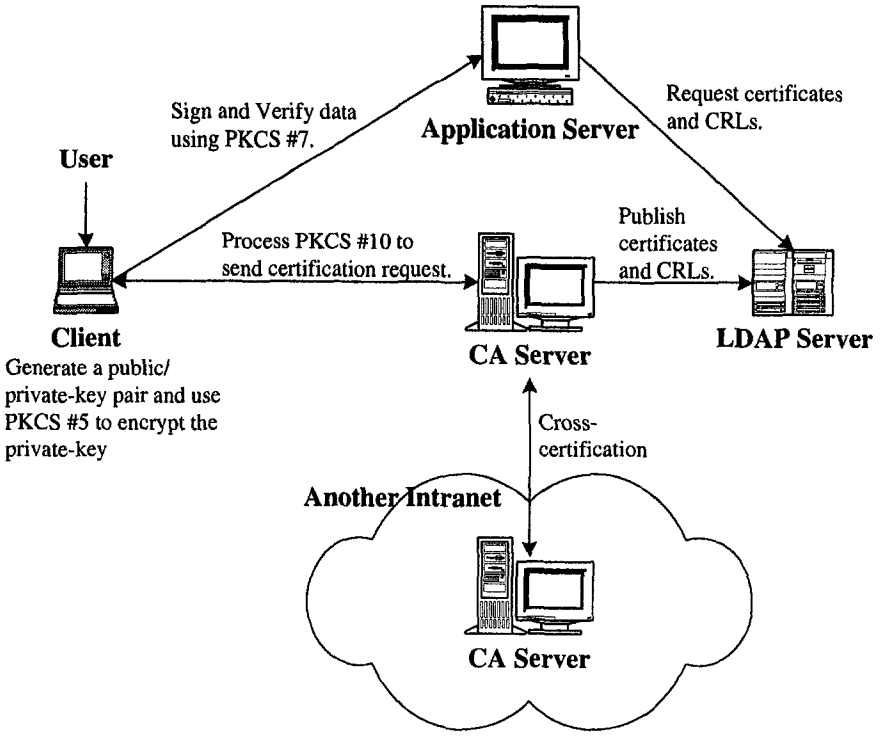


Figure 3.4: PKI system.

3.3.3 The System Design

In this section, considerations for designing a PKI system are described. A PKI generally consists of CA servers, directory servers, and client modules. The proposed PKI is to be constructed in the Intranet, in which the SSO has been implemented, so we design a PKI consisting of one CA server, one directory server, and multiple clients. Figure 3.4 shows the PKI system. It is shown that the CA server is in charge of one SSO-implemented Intranet and cross-certifies another CA servers. We mentioned the structures of CA servers for cross-certification in section 3.3.2, but its implementation is beyond the scope of this dissertation.

The main role of the PKI may be to generate and issue certificates for its users. The

procedure for generating a certificate for RSA cryptosystem is roughly described as:

Step 1. A user generates an RSA key pair according to PKCS #1 and encrypts the private-key with his own password according to PKCS #5 and PKCS #8;

Step 2. The user prepares a PKCS #10 certification request and sends it to the CA server;

Step 3. The CA server generates a certificate using the given certification request and saves the certificate into the LDAP directory. Optionally, the CA server sends the certificate to the user according to PKCS #7 signedData content type.

The CA server allows a user to send a certification request and construct a certificate from it. The CA server performs the followings:

- generates certificates from the certification request;
- generates and verifies digital signatures;
- accesses the directory server using LDAP.

The PKI makes use of directory services based on LDAP. The directory provides a means for storing and obtaining the certificates of all users including CA as well as CRLs. The LDAP server performs the following functions:

- stores the certificates and CRLs;
- makes all stored certificates available on request.

PKI clients use the public-key certificates that are issued and managed by the PKI to provide security services to users. The PKI clients perform the following functions with and for certificates:

- generate public/private-key pairs;
- generate certification requests;
- generate and verify digital signatures;

More details about designing and implementing the PKI will be given in Chapter 5 and the proposed PKI will also be implemented.

Chapter 4

Design of Single Sign-On

4.1 Single Sign-On Overview

Intranet users are commonly required to use separate password to authenticate themselves to each server they need to access. Multiple passwords are ongoing trouble for both users and system administrators.

Users have difficulty keeping different passwords in mind, so tend to choose bad passwords, write them down in obvious places, or choose only one password for all servers. System administrators must also keep track of a separate password database on each server and deal with potential security problems related to the fact that passwords are sent over the network frequently.

Solving this problem requires some way for a user to log in once using a single password and get authorized access to all servers that the user is authorized to access, moreover without transmitting any password over the network. If it is, then it is possible for users to select one complicated password hard to guess and crack. This concept is known as SSO. The SSO system has several benefits for both users and system administrators:

- Ease of use: Users can log in once and get authenticated access to all servers for which that user is set up to be authorized, without being interrupted by repeated requests for passwords;

- Password limited to local machine: To log in, the user types a single password that protects the cryptographic key database on the local machine. Passwords are not sent over the network;

- Access control not affected: SSO replaces only client authentication mechanism, not access control mechanism. System administrators do not need to change existing access control lists that may have been originally set up to work with basic password authentication.

In this chapter a SSO system based on a PKI is proposed. The proposed system involves the use of digital signature and cryptographic authentication protocols to authenticate users to server. Moreover, it provides the mechanism for preventing the masquerade attack that a host impersonates the identity of the server.

4.2 Previous Works

In this section two remarkable systems on the implementation of SSO are described: Kerberos system which is based on secret-key cryptosystem and Netscape's SSO based on public-key cryptosystem.

4.2.1 Kerberos System

Kerberos is an authentication system developed as part of Project Athena at MIT. Rather than building in authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users, that is, to perform mutual authentication. Unlike most other authentication systems, Kerberos implements SSO using secret-key cryptosystem. The Kerberos model is based on Needham-Schroeder's trusted third-party protocol [50]. Two versions of Kerberos exist: Version 4 [51, 52] is still in common use, and version 5 [53] complements security deficiencies of version 4 and has been issued as a draft Internet Standard.

To perform client authentication, each server must be able to confirm the identities of users who request service. Each server can be required to undertake this task for each client/server interaction, but this impose a substantial burden on each server. An alternative employed in Kerberos is to use an authentication server(AS) that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret-key with each server.

To avoid plaintext transmission of the password, Kerberos introduces a new server known as the ticket-granting server(TGS) which issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket(TGT: Ticket_{TGS}) from the AS. This ticket is saved in the client(C). Each time the user requires access to a new service(S), the client applies to the TGS using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket(SGT: Ticket_S) and uses it to authenticate its user to a server each time a particular service is requested.

In the simplified protocol specification for Kerberos, a normally executed authenti-

cation procedure begins with the exchange of the following five messages:

1. Once per user log-in session:
   - C $\rightarrow$ AS: $ID_C \parallel ID_{TGS}$;
   - AS $\rightarrow$ C: $E_{K_C}[Ticket_{TGS}]$;
2. Once per type of service:
   - C $\rightarrow$ TGS: $ID_C \parallel ID_S \parallel Ticket_{TGS}$;
   - TGS $\rightarrow$ C: $Ticket_S$;
3. Once per service session:
   - C $\rightarrow$ S: $ID_C \parallel Ticket_S$,

where ID_C and ID_{TGS} mean the identifier of the client and the TGS, respectively.

The TGT is encrypted with a key that is derived from the user's password. Thus, we know that the client uses the password to obtain credentials from the AS without having to transmit the password in plaintext. Two tickets, $Ticket_{TGS}$ and $Ticket_S$, are expressed as

$$Ticket_{TGS} : E_{K_{TGS}}[ID_C \parallel AD_C \parallel ID_{TGS} \parallel TS_1 \parallel Lifetime_1]$$

$$Ticket_S : E_{K_S}[ID_C \parallel AD_C \parallel ID_S \parallel TS_2 \parallel Lifetime_2],$$

where AD_C is the network address of the client, and the timestamp(TS) and lifetime(Lifetime) are used to indicate whether the ticket is valid.

This scheme satisfies the two requirements of only one password query and protection of the password. Figure 4.1 provides a simplified overview of Kerberos system.

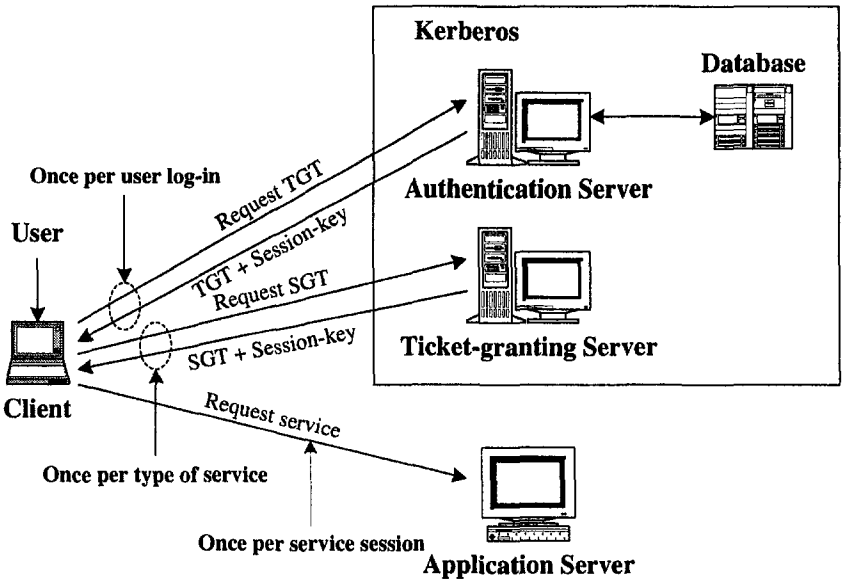


Figure 4.1: Overview of the Kerberos system.

4.2.2 Netscape Single Sign-On

Netscape's approach to SSO replaces client authentication based on passwords sent over the network with that based on the SSL and public-key certificates.

The SSL standard supported by Netscape addresses the need for authentication, privacy, and data integrity. SSL is a protocol that runs above TCP/IP and below HTTP, LDAP, IMAP, NNTP, and other high-level network protocols to provide a reliable end–to–end secure service. SSL allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

A server-authenticated SSL connection makes it extremely difficult to eavesdrop on the connection, modify the data without detection, or impersonate the identity of the server. However, unless the client as well as the server is authenticated, any user can establish a connection and gain access to the resources managed by the server.

In Netscape SSO system, a single certificate rather than multiple passwords to authenticate a client to multiple servers. To authenticate a user to a server, the client digitally signs a randomly generated data and sends both the certificate and the signed data in the SSL payload.

Netscape's authentication process requires the use of SSL and takes places after the initial server authentication. It is also assumed that the client has a valid certificate that can be used to identify the client to the server.

The SSO system proposed by Netscape is demonstrated in Figure 4.2 and has the following steps:

Step 1. The client software, Netscape Communicator or Navigator 3.x, maintains a database of the private-keys that correspond to the public-keys published in any certificates issued for that client. The client asks for the password to this database the first time the client needs to access it.

Step 2. The client unlocks the private-key database, retrieves the private-key for the user's certificate, and uses that private-key to digitally sign some data that has

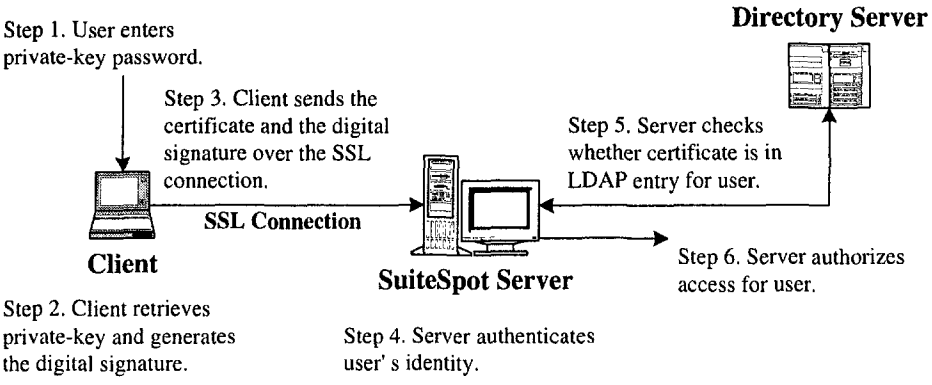


Figure 4.2: Netscape's SSO system.

been randomly generated for this purpose on the basis of input from both the client and the server.

Step 3. The client sends both the user's certificate and the digitally signed data over the SSL connection.

Step 4. The server uses the certificate and the digitally signed data to authenticate the user's identity.

Step 5. The server maps the user's identity to a unique entry in the LDAP directory and checks that the entry contains the same certificate that was sent to the server.

Step 6. If the LDAP lookup is successful, the server allows the user to access it according to access control lists.

4.3 The Proposed Single Sign-On

4.3.1 The System Overview

Client authentication is an essential element of network security within most Intranets or Extranets. Almost all server softwares permit the client authentication by means of an ID and password. This information may be sent in the clear or over a server-authenticated SSL connection. However, even though the password is encrypted over the SSL connection, it is still possible to capture the encrypted password packet and retransmit it subsequently to get an authenticated access.

The proposed SSO system consists of an authentication server, LDAP server, multiple application servers and clients. The system diagram is shown in Figure 4.3 and an

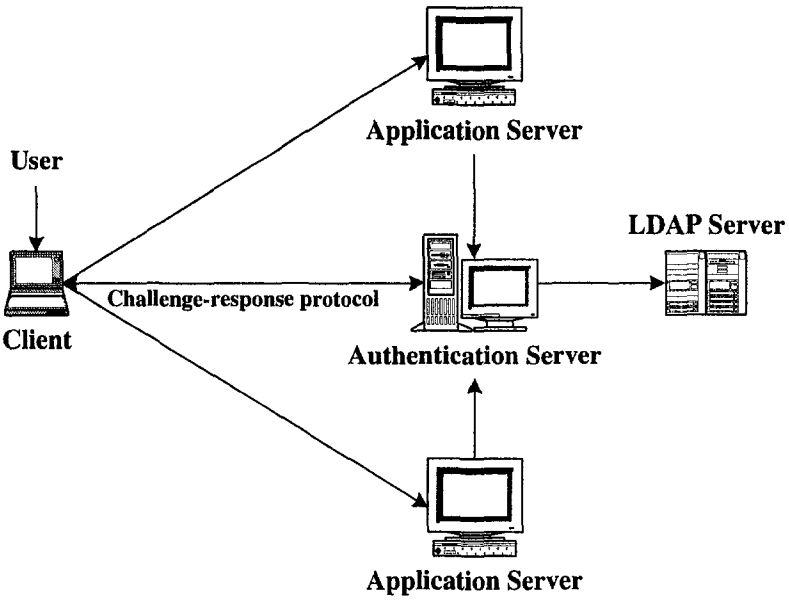


Figure 4.3: The proposed SSO system.

overview is roughly sketched as follows:

Step 1. A user accesses an application server and inputs the electronic mail address(e-mail) as an identifier;

Step 2. The application server forwards the e-mail to the authentication server with its domain-name and a random string of numbers;

Step 3. The authentication server checks whether the owner of the e-mail is being in authenticated state or not. If he is, go to Step 5. Otherwise, go to the next step;

Step 4. The authentication server performs a cryptographic authentication protocol with the client. If the authentication process is performed successfully, go to the next step. Otherwise, perform the authentication process repeatedly. The details of the authentication process is described in Section 4.3.2;

Step 5. The authentication server generates a ticket by encrypting the previously received random string with the secret-key that both the application server and the authentication server has previously shared. It sends the ticket to the application server with the e-mail;

Step 6. The application server decrypts the given ticket with the secret-key and compares it with the original random string. If they match, the user is allowed to use the application server as well as other application servers by simply typing the e-mail address.

4.3.2 Technique for Performing an Authentication Protocol

The proposed SSO system uses the RSA digital signature algorithm for challenge-response authentication protocol. The procedure for implementing the authentication protocol is described as:

Step 1. The client maintains a database of the private-keys encrypted with each user's password. The client asks for the password to this database to load the private-key. After entering this password once, the user does not need to enter it again for the rest of the session, even when accessing other application servers;

Step 2. The client unlocks the private-key database, retrieves the private-key for the user's certificate, and uses that private-key to digitally sign the challenge data that has been randomly generated and sent by the authentication server. RSA digital signature algorithm is employed to generate the signature;

Step 3. The client sends the digitally signed data to the authentication server over the network;

Step 4. The authentication server maps the user's e-mail address to a unique entry in the LDAP directory and checks that the entry contains a certificate. This step assumes that the CA server has been configured to publish each certificate it issued in the directory;

Step 5. If the certificate is in the directory, the authentication server fetches the certificate to verify the signed data. Otherwise, the authentication process fails.

If we look over only the authentication protocol, the proposed protocol seems to be analogous to the Netscape's SSO. But there are some differences: In Netscape's scheme, a user who wants to access a particular application server should connect a centralized authentication server called SuiteSpot server first of all. But in the proposed system, a user simply connects the wanted application server, which automatically requests authentication service from the authentication server. And that, as mentioned above, the proposed system is independent of the use of SSL. The Netscape's scheme sends the authentication information, which is the certificate and the digitally signed data, in the SSL payload. However, the proposed system is implemented fully above the HTTP layer.

4.3.3 Requirements Capture of the Proposed System

In this section, run of the proposed system has been viewed. Message Sequence Chart(MSC) is employed for visualization of the system runs within the Intranet in which SSO has been implemented [54]. MSCs have been used informally for a long time by ITU Study Groups in their recommendations and in the industry. MSC can be viewed as a special trace language which mainly concentrates on message exchanges by communicating entities and their environment. A main advantage of MSC is its clear graphical layout

Message	Parameters	Subjects
service_req	URI, userEmail	Client(User)
service_resp	flag	↔ Application Server
auth_req	serverName, userEmail, randStr	Application Server
auth_resp	userEmail, *ticket*	↔ Authentication Server
password_req	–	Authentication Server
password_resp	flag	↔ Client(User)
cert_req	–	Authentication Server
cert_resp	certificate	↔ LDAP Server
signature_req	randNum	Authentication Server
signature_resp	signedData	↔ Client(User)

Table 4.1: Messages exchanged between system components: – means that the message has no parameters.

which immediately gives an intuitive understanding of the described system behavior.

A number of messages are defined to describe the behavior of the components comprised in the proposed system. Table 4.1 shows the messages and their parameters. Service_resp and password_resp messages has flag parameter, whose value is success(S) or failure(F), and the values of other parameters can easily be guessed from the name of parameters.

We introduce a number of scenarios which can happen in the real deployment and present the MSC on each scenario, to correctly design the system.

Scenario 1

This scenario represents the general case in which a legitimate user connects an application server for the first time. MSC for this scenario is illustrated in Figure 4.4.

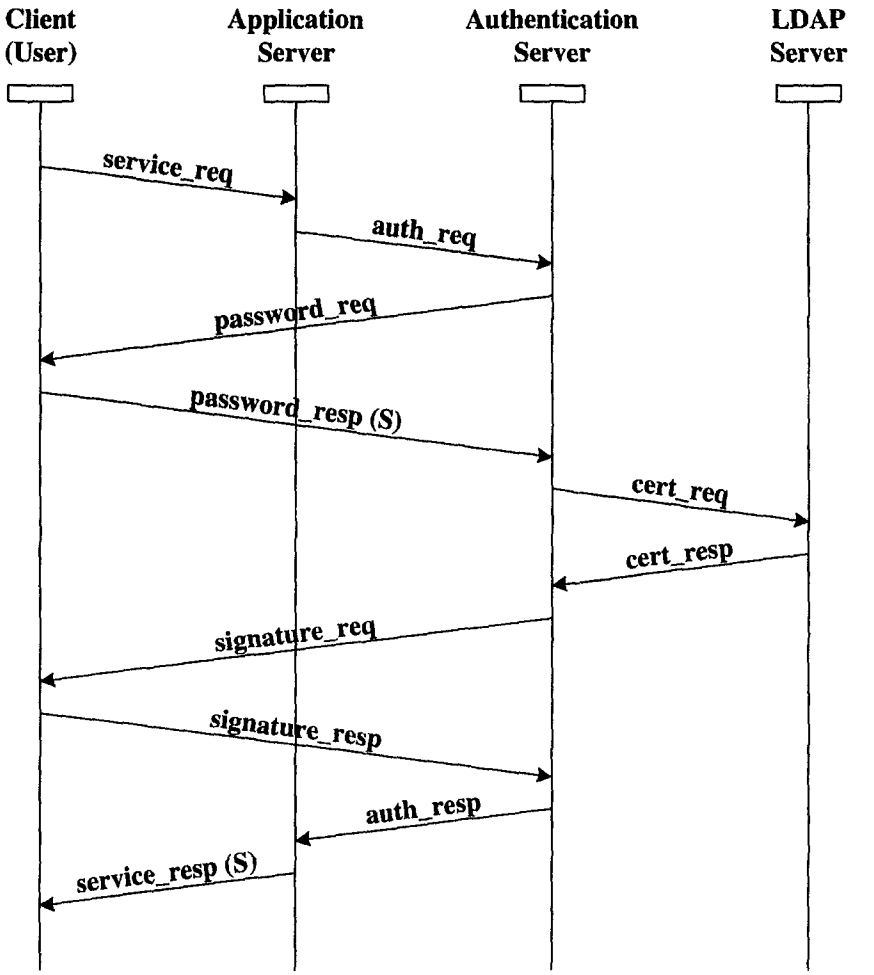


Figure 4.4: Scenario 1: General case.

The procedure for exchanging messages is as follows: A client sends service_req message with the user's e-mail(userEmail) to an application server. The application server requests the client authentication from the authentication server by sending auth_req message with parameters of the application server's domain-name(serverName), userE-mail, and random string(randStr). The randStr is employed to prevent an opponent from directly accessing an application server without performing authentication protocol.

The authentication server asks the user for the password for the private-key database

by sending password_req message. The client receives the password from the user to retrieve the private-key. If this succeeds, the client responds to the authentication server by sending password_resp message with S flag. The authentication server requests the user's certificate from LDAP server. If both the entry and the certificate exist in the LDAP directory, the LDAP server sends the certificate as cert_resp message. Then the authentication server performs authentication protocol with the client. Signature_req and signature_resp messages are used in the authentication process. The details of authentication protocol are described in Section 4.3.2.

After the authentication process ends successfully, the authentication server responds to the application server by sending auth_resp message with parameters of userEmail and ticket. The ticket and the randStr work in pairs. Note that the ticket is encrypted with the secret-key known only to the application server and the authentication server. Therefore, impersonating an application server seems to be impossible because a disguising host cannot decrypt the ticket.

After the application server checks the ticket, if the ticket is considered as valid, then the application server responds to the client by sending service_resp message with S flag.

Scenario 2

This scenario is the same as the scenario 1, except that user enters wrong password. The client given wrong password from the user fails to retrieve the user's private-key, then responds to the authentication server by sending password_resp message with F flag. The authentication server asks for right password repeatedly. MSC for this scenario is illustrated in Figure 4.5.

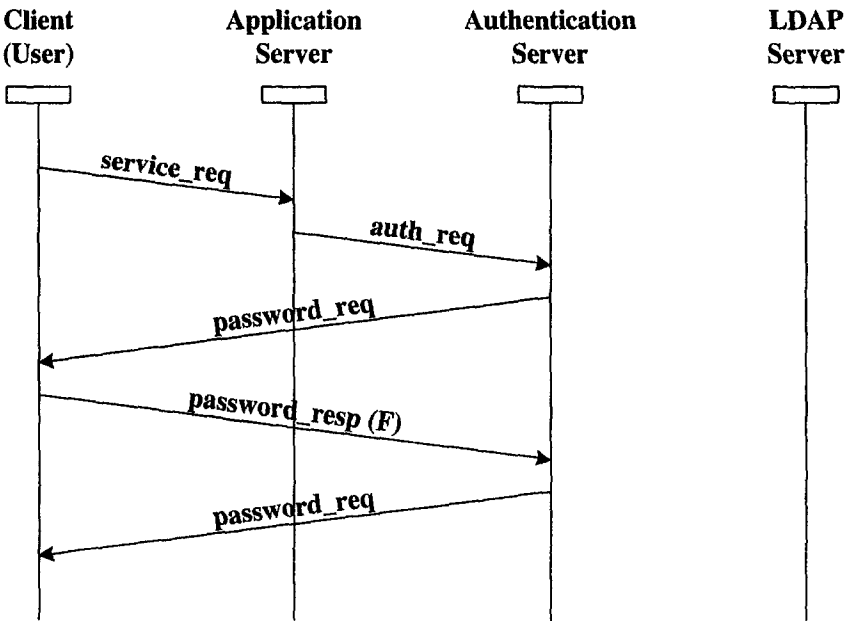


Figure 4.5: Scenario 2: In case a user enters invalid password.

Scenario 3

System administrator can manage the access control lists of all users using LDAP directory server. CA server publishes each certificate it issues in the LDAP directory. To disallow authentication for a particular user, the administrator simply removes the user's certificate from the LDAP directory. This action prevents the user from accessing any of the servers, even if the user's certificate has not expired.

This scenario presents the case in which LDAP directory lookup fails. In this case, neither signature_req nor signature_resp messages are sent. Authentication server sends to the application server auth_resp message including dummy ticket(e.g., all 0's bit-stream). The application server given dummy ticket responds to the client by sending service_resp message with F flag. MSC for this scenario is illustrated in Figure 4.6.

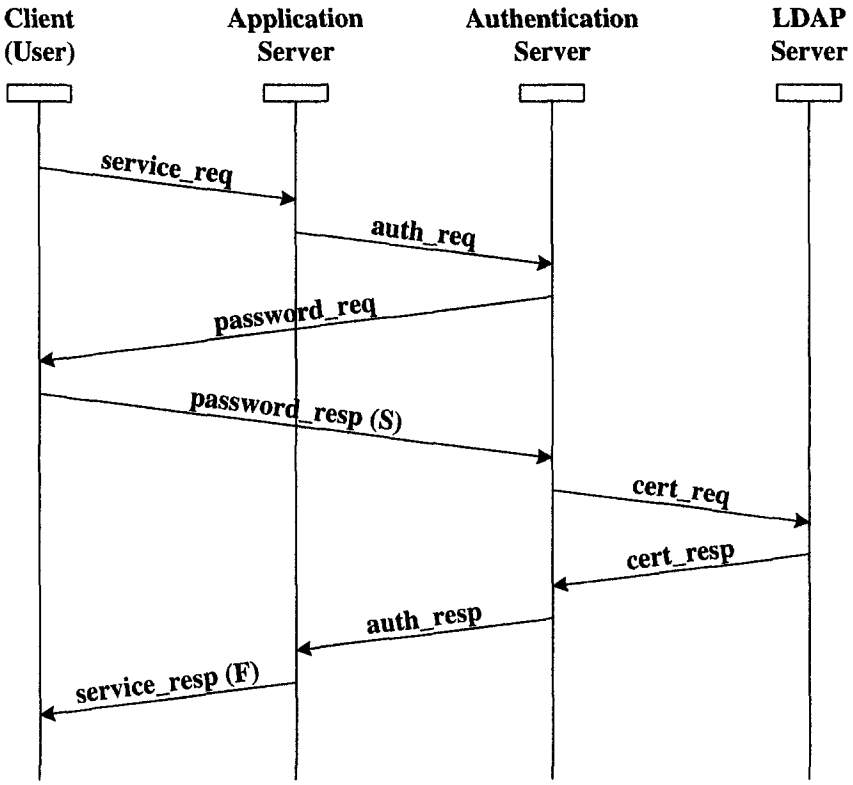


Figure 4.6: Scenario 3: In case the LDAP directory lookup fails.

Scenario 4

This scenario assumes the case in which the signature generated by the client has been modified on transmission. If verification of the signedData fails, it does the same operation as the scenario 3. MSC for this scenario is illustrated in Figure 4.7.

Scenario 5

This scenario assumes that a user has been authenticated according to the scenario 1. Note that the authentication server sets a timer(T_{Auth}), on sending auth_resp message to the application server.

Let us assume that the user wants to access the same application server from the

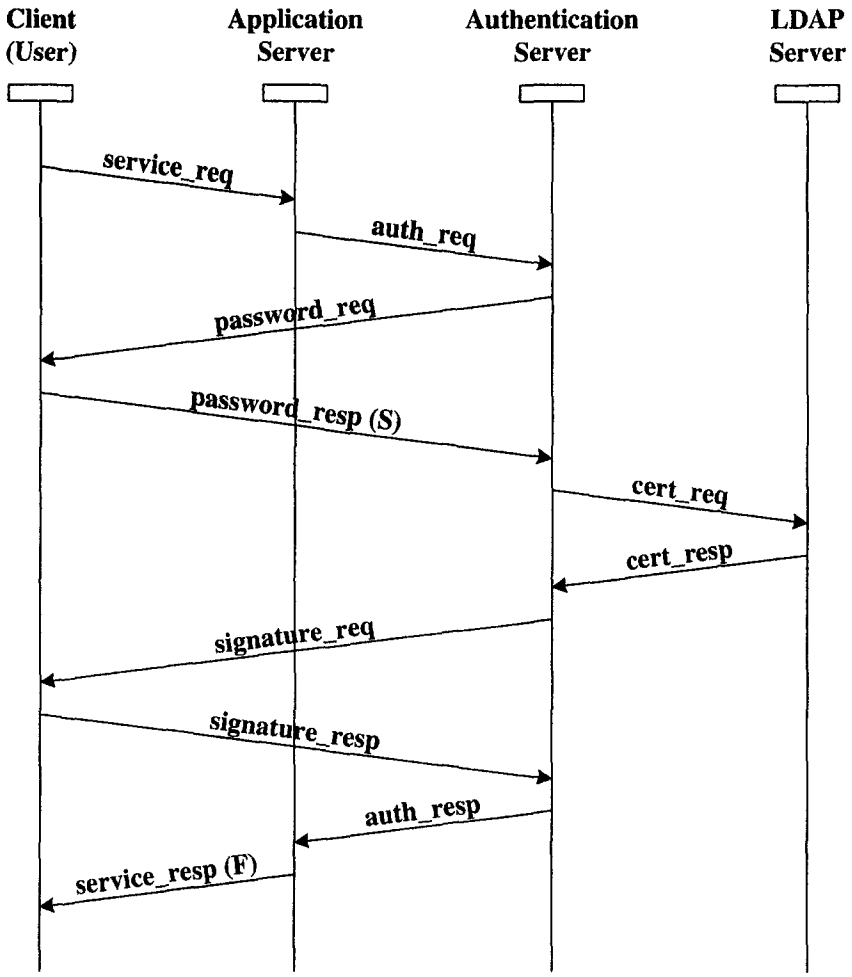


Figure 4.7: Scenario 4: In case the signature has been modified on transmission.

same client again. If the user accesses the application server before the timer expires, he does not need to be authenticated; The user can access the application server without performing authentication protocol.

There is one thing to mention about the lifetime of the timer. If this lifetime is very short(e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long(e.g., hours), then an opponent has a greater opportunity for attack. An opponent waits for the legitimate user to log out and then access the application server

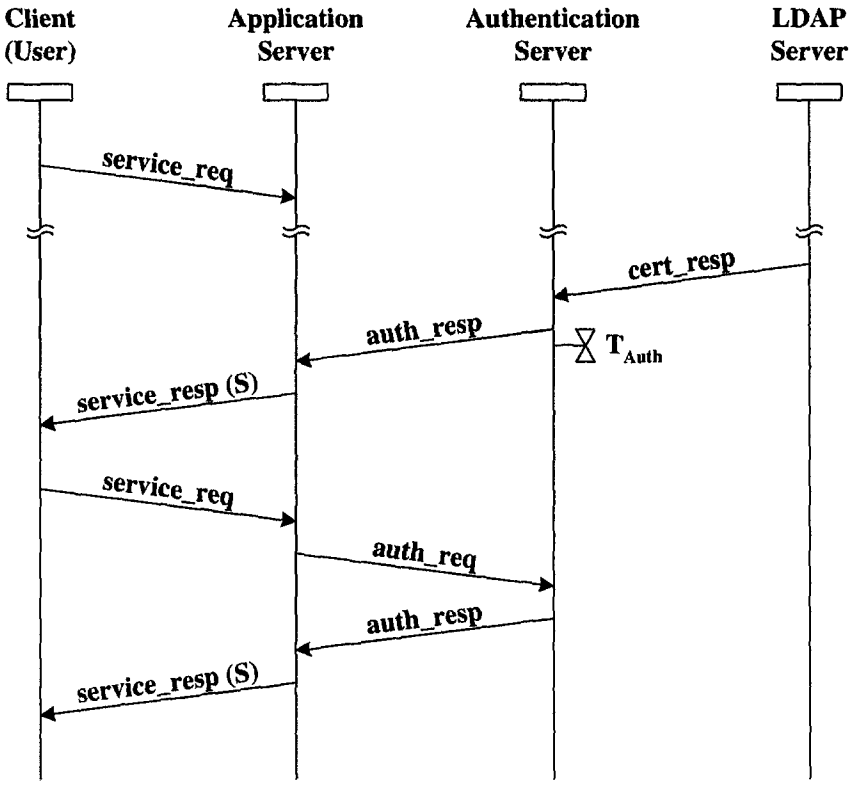


Figure 4.8: Scenario 5: In case a user accesses the same application server from the same client within timer expiration.

only by entering the legitimate user's e-mail. Then the opponent can use unlimitedly the application server without authentication process. For the prevention of this attack, a log-out mechanism is introduced. When a user logs out, the user's log-in information is removed from the authentication server. MSC for this scenario is illustrated in Figure 4.8.

Scenario 6

This scenario is the same as the scenario 5, except that the user accesses the application server after the timer(T_{Auth}) expires. In this case, the authentication server does not

have the user's log-in information, so the authentication process must be performed according to the scenario 1. MSC for this scenario is illustrated in Figure 4.9.

Scenario 7

In this scenario, a user who has been authenticated to application server 1 wants to access another server denoted by application server 2. Note that the timer(T_{Auth}) was set in the first authentication process. Only if the user accesses the application server 2 within timer expiration, the user will be authenticated without being asked for entering the password. It is worth noting that the authentication server manages all accesses to application servers with its timer. MSC for this scenario is illustrated in Figure 4.10.

Scenario 8

This scenario is the same as the scenario 7 except that the user accesses the application server 2 after the timer(T_{Auth}) expires. As mentioned in the scenario 7, the authentication server sets its timer when the first log-in succeeds, and checks the timer on each log-in. Because the timer expires, client must be authenticated according to authentication process explained in the scenario 1. MSC for this scenario is illustrated in Figure 4.11.

Scenario 9

This scenario assumes that a user has been authenticated according to the scenario 1. The authentication server sets a timer(T_{Auth}), on sending auth_resp message to application server.

There is some possibility that an opponent may access the application server using the legitimate user's e-mail from another client. To prevent this type of attack, authen-

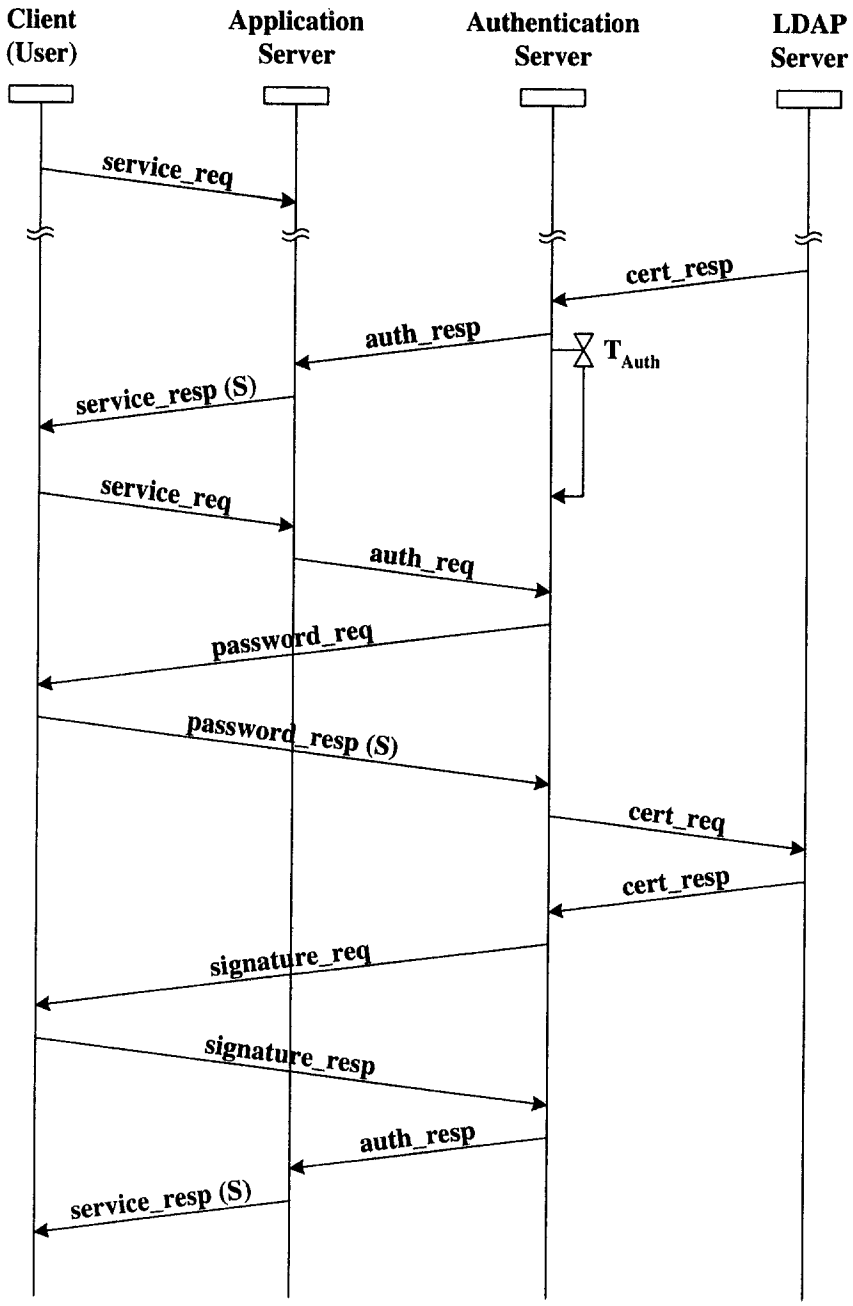


Figure 4.9: Scenario 6: In case a user accesses the same application server from the same client after the timer expires.

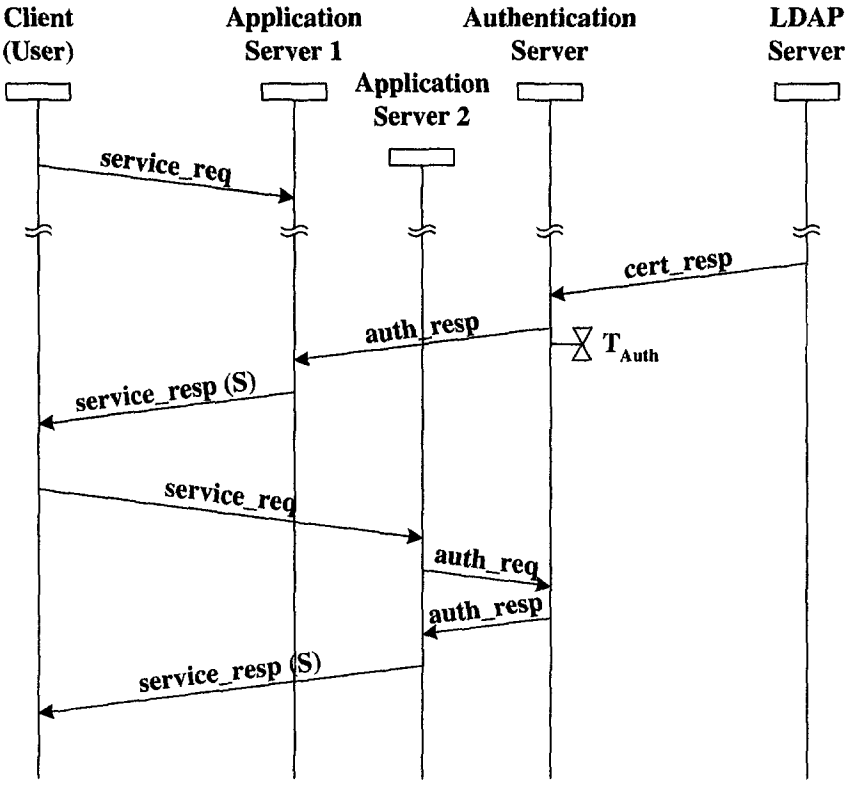


Figure 4.10: Scenario 7: In case a user accesses another application server from the same client within timer expiration.

tication server checks the network address of the client.

When the authentication server sends auth_resp message to application server to complete authentication process successfully, it also saves the network address of the client into its own database. Later the authentication server receives auth_req message, it looks up the database. If the two network addresses do not match, then even if the access is requested within timer expiration, the opponent must perform authentication process. Because the opponent do not know the password, he cannot respond to password_req message, so cannot access the application server. MSC for this scenario is illustrated in Figure 4.12.

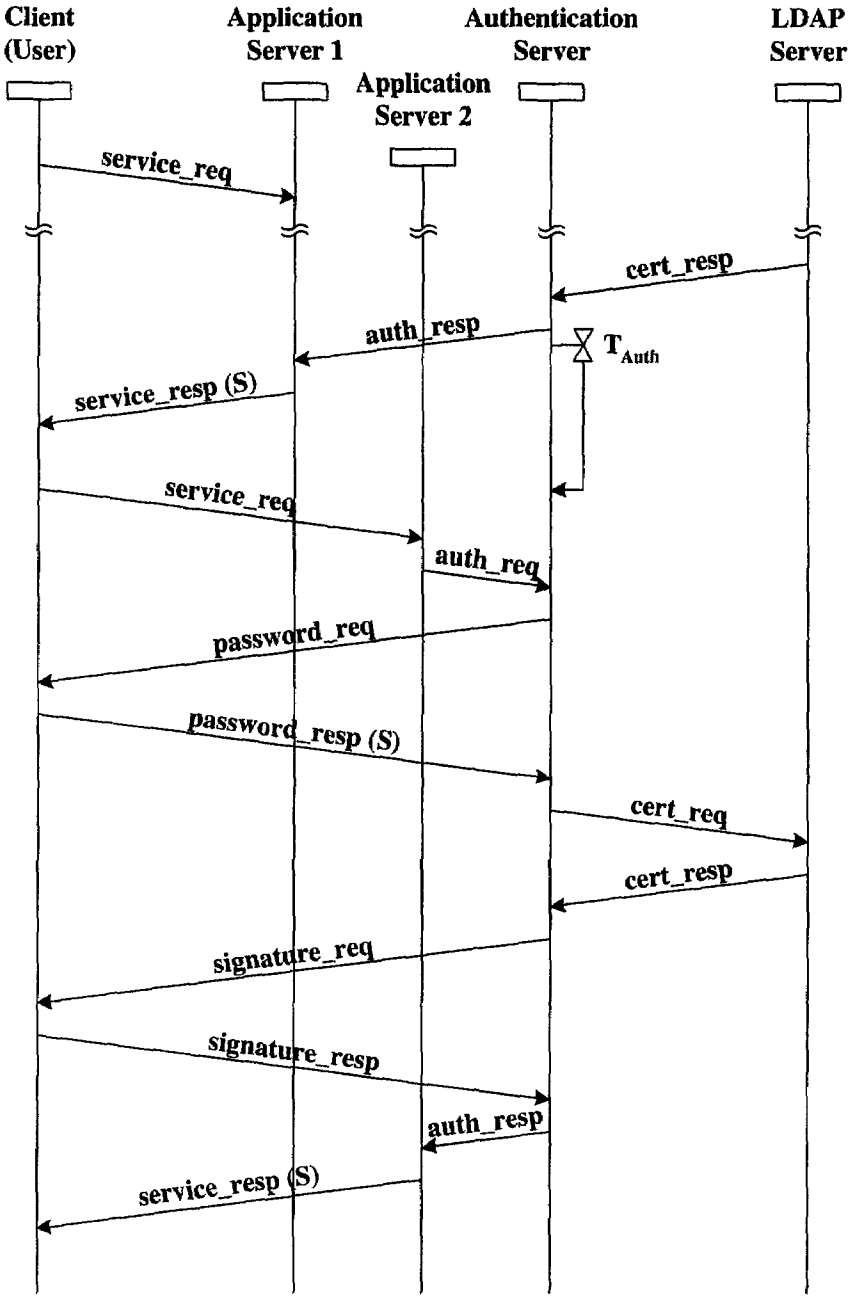


Figure 4.11: Scenario 8: In case a user accesses another application server from the same client after the timer expires.

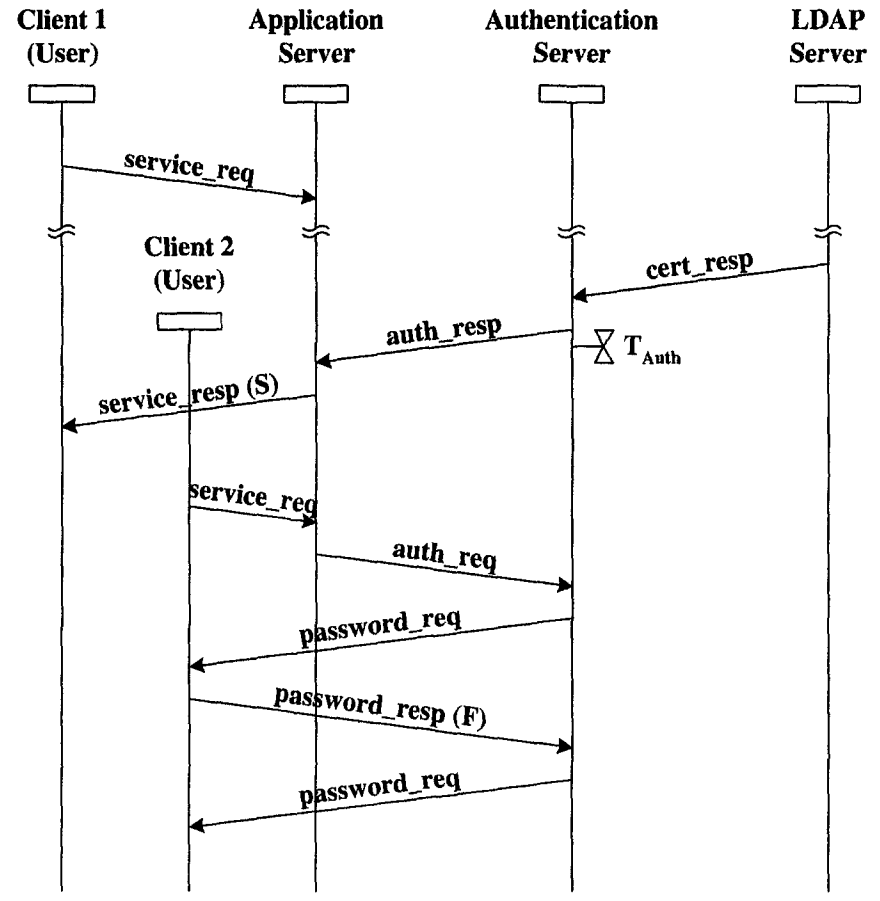


Figure 4.12: Scenario 9: In case an opponent accesses the application server from the another client within timer expiration.

Scenario 10

In this scenario, an opponent may directly access an application server without authentication. To prevent this attack, the randStr and the ticket are employed.

Note that service_req message has the uniform resource identifier(URI) parameter. Application server can be accessed through two URIs: One is for the first access, and the other is for the access with auth_resp message. An opponent can directly access an application server through the latter URI. But the directly accessing opponent cannot

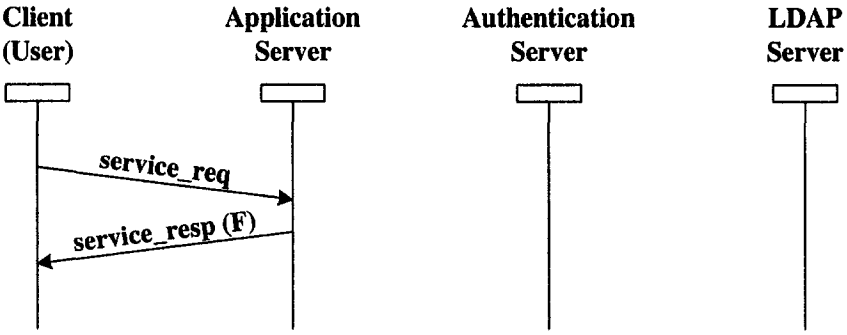


Figure 4.13: Scenario 10: In case an opponent directly accesses an application server without authentication.

generate a ticket which is necessary for authentication. Even if he captured the ticket in the previous transmission, its retransmission cannot be considered as valid, because a ticket is generated by encrypting the randStr which is generated randomly in each authentication process. MSC for this scenario is illustrated in Figure 4.13.

Scenario 11

The proposed system uses a reliable transport layer such as TCP, but network packets generated in the authentication process can get lost. Users tend to retransmit the data, unless the response arrives from the server within reasonable time. So, in case there are some network problems, a number of service_req messages may be sent to an application server.

Application server handles this problem by setting a timer(T_{App}) on sending auth_req message to the authentication server. See Figure 4.14. If a service_req message arrives within timer expiration, the application server ignores the message. But if a service_req message arrives after the timer expires, the new service_req message is accepted and processed. Even though service_resp message to the previous service_req message arrives, the response message will be ignored. MSC for this scenario is illustrated in

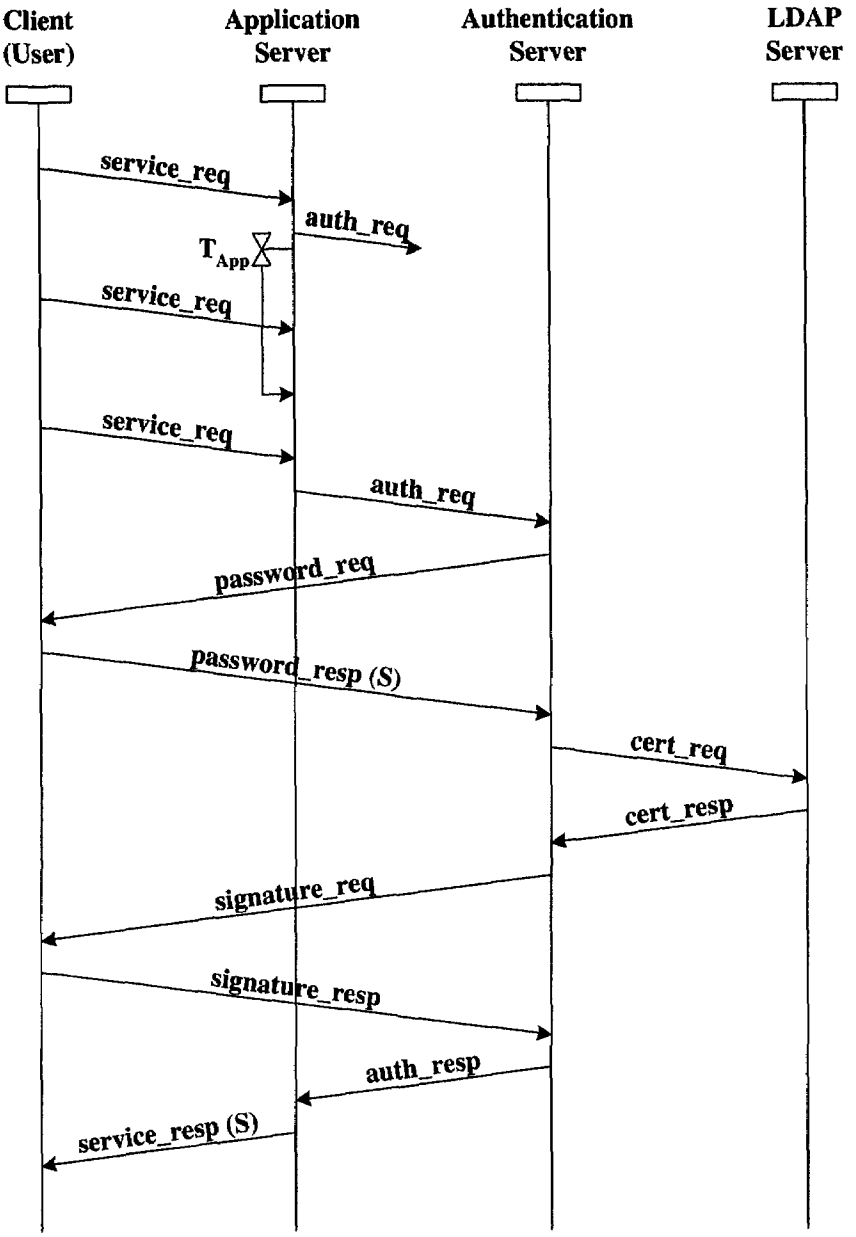


Figure 4.14: Scenario 11: In case a user sends multiple service_req messages.

Figure 4.14.

Chapter 5

Implementation of Single Sign-On

5.1 Total System Overview

In this chapter, the proposed SSO system has been implemented and discussed. The proposed SSO system is based on the implementation of PKI and the use of digital signature and is intended to be implemented on the Intranet. The reason we limit the implementation environment to the Intranet is that all application servers should have the same interface to both the CA server and the authentication server. But, it can easily be extended to the Internet environment.

For the proposed system to properly operate, a PKI should be first constructed. The proposed PKI is to be constructed in the Intranet environment, so it would include one CA server, one directory server, and several clients.

After constructing the PKI, the SSO system is implemented. The proposed SSO system consists of one authentication server, two application servers, one directory server, and several clients as well as a PKI. The total SSO system including the PKI is shown in Figure 5.1.

The proposed SSO is to be implemented in the Intranet in which most services are

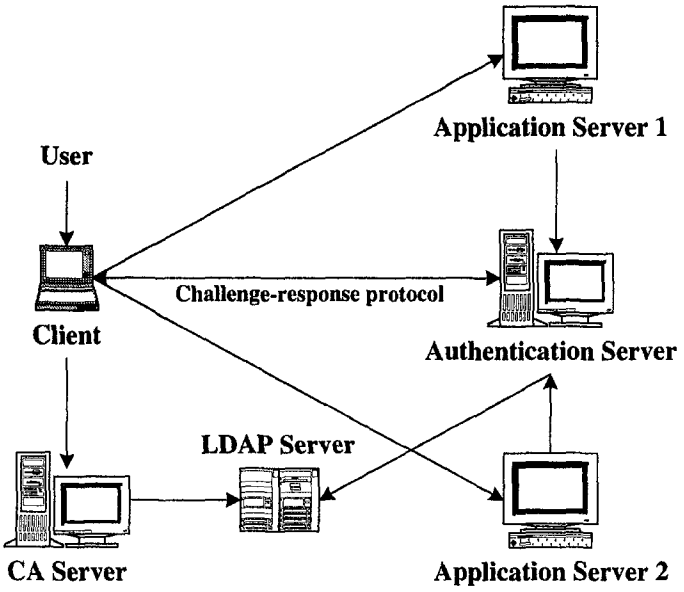


Figure 5.1: The proposed and implemented single sign-on system.

provided by WWW servers. Therefore, all servers provide WWW services and are developed on Linux platform. To build a WWW server, we employ an open-source server called Apache HTTP server, which has been the most popular web server on the Internet. It is known that over 60% of the web sites on the Internet are using Apache server, thus making it more widely used than all other web servers combined. Table 5.1 shows the implementation environment of the proposed system.

5.2 Implementation of Public-Key Infrastructure

5.2.1 Implementation of CA Server

The proposed CA server performs the following four functions:

- generates certificates;

Function	CPU(MHz)	Operating System
CA Server	Pentium-III 500	RedHat Linux 6.1
LDAP Server	Pentium-III 450	RedHat Linux 6.1
Authentication Server	Pentium-III 500	RedHat Linux 6.1
Application Server 1	Pentium-III 450	RedHat Linux 6.1
Application Server 2	Alpha Axp 533	RedHat Linux 6.1
Client	Pentium-II 400	Microsoft Windows 98

Table 5.1: Implementation environment.

- registers the certificate into the LDAP directory;
- issues certificates;
- revokes certificates.

Figure 5.2 shows the initial page for a user's applying a certificate to the CA server.

First, the CA server receives the certificate request from a client, and then fulfills the request by authenticating the requesting user and verifying the user's signature, and if the request is valid, constructing an X.509 certificate from the name object and public-key, the issuer name, the CA's choice of serial number, validity period, and signature algorithm. The format of X.509 certificate is in detail described in Section 3.2.1. The CA server generates a certificate using the values described in Table 5.2.

Then, the CA server registers the generated certificate into the LDAP directory. A LDAP server generally allows ASCII-formatted data in its directory, but a certificate is presented as a octet string. For converting between binary and ASCII stream, base-64 encoding/decoding algorithm [18] is employed. Details about implementation of the LDAP server are described in Section 5.2.2.

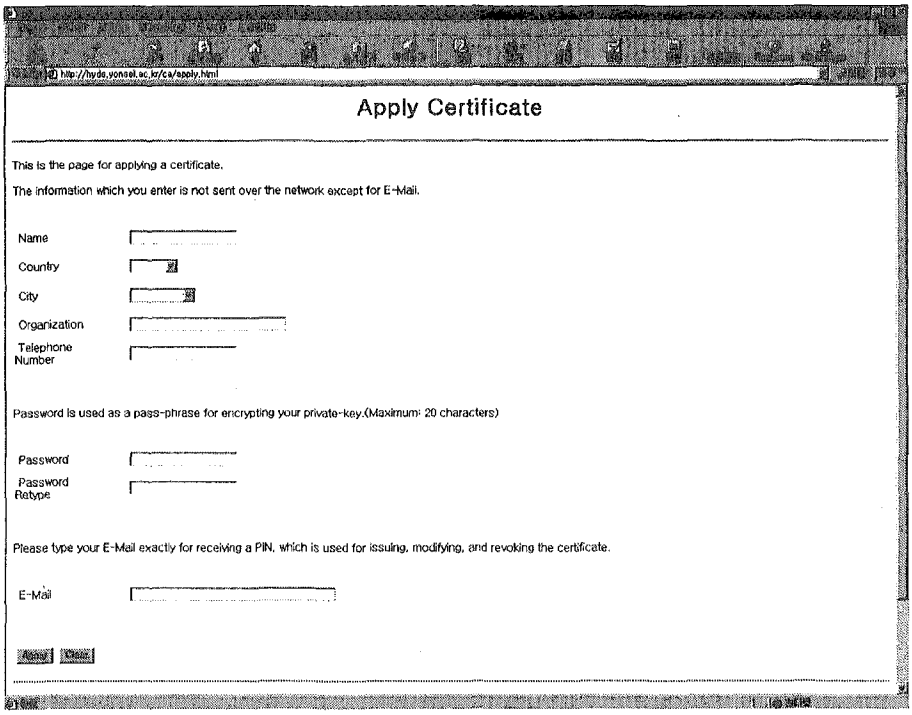


Figure 5.2: The initial page of the CA server.

The CA server also provides the function to issue the certificate to the user. This issuing function is not a requirement of CA server, but optional provision. To issue the certificate, a data format, in which a certificate is sent, must be defined. PKCS #7,

Element	Value
Version	ver. 1
Signature algorithm identifier: Algorithms	MD5 with RSA encryption
Issuer name	Root CA(Yonsei CA)
Period of validity: Not before	present date
Period of validity: Not after	present date + 1 year
Subject's public-key info: Algorithms	RSA

Table 5.2: The values used in the X.509 certificate generated by the proposed CA server.

a general syntax for data to which cryptography may be applied, defines six content types: data, signedData, envelopedData, signedAndEnvelopedData, digestedData, and encryptedData. Note that the CA's certificate as well as the user's certificate is needed for authentication processes, because the user's certificate can be verified only by the CA's certificate. The CA server generates PKCS #7 signedData message consisting of both the CA's and the user's certificate and distributes it to the user.

For a user to download a certificate, he should be authorized as the owner of the certificate. For this implementation, a personal identification number(PIN) is introduced. After constructing a certificate, the CA server generates a PIN for the user using the random number generator. Then, the CA server saves the PIN into the local database with the user's e-mail and sends the PIN to the user via e-mail. The user who wants to obtain the certificate accesses the certificate-issuing page of the CA server, and can download the certificate by entering the given PIN. The MySQL database management system, open-source server, is used for local database.

The CA server simply deletes a user entry and its attribute, which is the user's certificate, from the LDAP directory in order to revoke the certificate. The implemented CA server does not support the CRL, because every server which wants to get a certificate is programmed to access and get it from the LDAP server. Therefore, a centralized access control can be performed using the LDAP directory.

To implement the CA server, Common Gateway Interface(CGI) programming technique is employed and C language is also used as a server script language.

5.2.2 Implementation of LDAP Server

In this section, a LDAP server is designed and implemented. Certificates are typically stored in a directory system, with applications using the LDAP to retrieve them as needed.

The directory provides a means for obtaining the certificates of clients and CAs themselves as well as CRLs. This allows clients to verify certification paths and that unexpired certificates have not been revoked. It also allows clients to use the recipient's public-key from his certificate to send an encrypted message without prior key negotiation.

In some cases directory servers may be maintained by the authorities which issue and manage certificates, but this need not be the case, and more general directories maintained for routing or location services may also provide directory information, including certificates.

In LDAP directory, data are generally stored as entries in tree-like structure. An entry has a DN to identify itself uniquely within the directory, as well as one or more attributes that describe the entry. For such a LDAP directory to be used in the PKI, a structure of the directory as well as attributes should first be defined.

Figure 5.3 shows the proposed structure of the directory to be used as a certificate repository. The proposed directory has the typical tree structures: country(c), organization(o), organizational unit(ou), and entries distinguishable by common name(cn). Each entry has an attribute, which is the certificate of the cn.

Non-ASCII values, or values that are too long to fit on a reasonably sized line are generally represented using a base-64 encoding, making entries human readable and easy to reconstruct if lost. This is applied to the proposed structure in which each entry

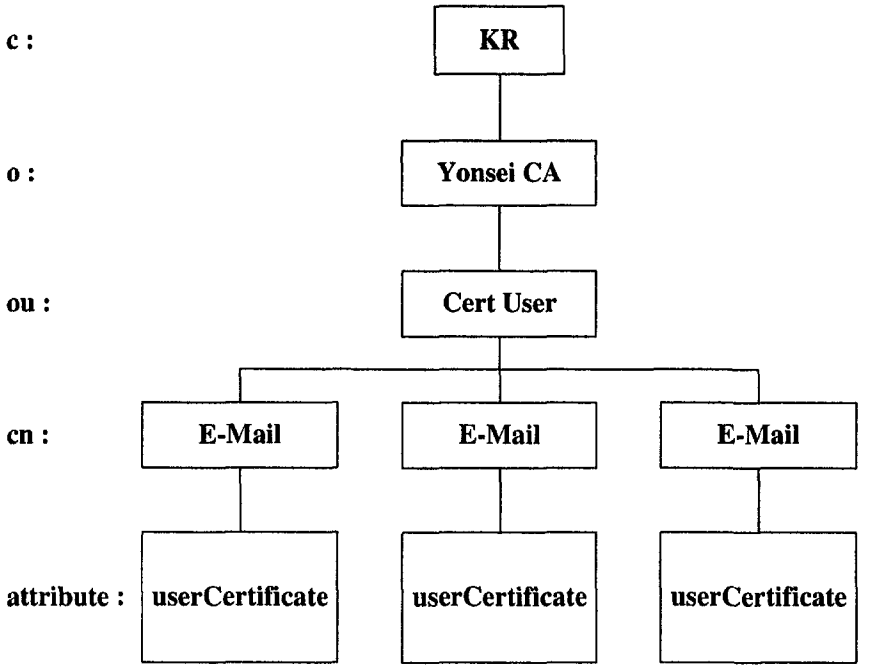


Figure 5.3: The proposed LDAP structure.

has a binary certificate as an attribute value.

First, We define a country. Since We are Korean, we set the country code to "KR". Then we define an organization as "Yonsei CA" and an organizational unit as "Cert User". Finally, each entry has each user's e-mail as a cn. In the proposed structure, for example, the DN of a cn "ysjeong@eve.yonsei.ac.kr" is defined as "dn: cn=ysjeong@eve.yonsei.ac.kr, ou=Cert User, o=Yonsei CA, c=KR". Each entry has an attribute value which is the user's certificate encoded into ASCII string by means of base-64 encoding.

To build an LDAP server, we employ a free open-source server called OpenLDAP. OpenLDAP release distribution provides a stand-alone LDAP daemon(SLAPD) server. SLAPD is an LDAP directory server that runs on many different UNIX platforms such as Linux. The configuration for running SLAPD as a local directory server with the

proposed structure is as follows [55]:

```
# slapd configuration
include     ./data/slapd.at.conf
include     ./data/slapd.oc.conf
schemacheck on
pidfile     ./CA-db/slapd.pid
argsfile    ./CA-db/slapd.args

# ldbm database definition
database    ldbm
cachesize   4
suffix      "o=Yonsei CA, c=KR"
directory   ./CA-db
rootdn      "cn=wmaster@hyde.yonsei.ac.kr, o=Yonsei CA, c=KR"
rootpw      passwd
index       cn,sn,uid  pres,eq,approx
index       default    none
lastmod     on
```

In the proposed LDAP server, two commands are used: "ldapmodify" and "ldapsearch". To store a certificate into the LDAP directory, it must first be converted into ASCII string using the base-64 encoding algorithm. Then, the CA server registers the certificate into the directory using ldapmodify command.

For the authentication server to perform authentication process based on cryptographic challenge-response protocol, the user's certificate should be obtained from the directory server. The authentication server checks whether the entry for the user and his certificate exist in the directory using the ldapsearch command. If this is, it fetches from the directory the certificate, which is in ASCII string format. Before used for authentication process, it must be decoded into a octet string.

5.2.3 Implementation of Client Module

A client module of PKI performs the following functions:

- generates public/private-key pairs;
- encrypts the private-key and saves into the local machine;
- generates certification requests;
- communicates with the CA server to send certification requests;

First, we generate a key pair, which consists of a private-key and its associated public-key, for RSA cryptosystem. The RSA algorithm relies on very large prime numbers, which are produced during key pair generation. This could be fairly time-consuming process. Determining the size of the modulus is an important in points of both security and implementation. The larger the modulus, the greater the security. Unfortunately, the larger the modulus, the longer it takes to generate a key pair and to encrypt and decrypt. The size of the modulus is determined as 1024 bits, which is considered as secure in most applications.

The public exponent, that is public-key, is usually chosen as one of two values: 3 or $2^{16}+1$. This choice leads to the result that the proportion of messages which are unconcealed by RSA encryption will generally be negligibly small, and hence unconcealed messages do not pose a threat to the security of RSA cryptosystem in practice [56]. Then, we generate the associated private-key using the pseudo-random number generator, which is based on the message digest algorithm such as SHA-1 [57].

Then we encrypt the private-key and save it into the local system. For encryption, DES in CBC mode is employed where the secret-key is derived from a user-given password. In order to generate a secret-key from a password that a user entered when he applied a certificate to the CA server, a password-based encryption algorithm, which is standardized as PKCS #5 by RSA, is used.

Usually, a password does not have enough random bits to qualify as a candidate for a key or even a random seed to generate a key. Therefore, the password-based encryption implementation not only uses the password, but also mixes in a random number, known as a salt, to create the key. Normally, the mixing is a message digest algorithm. Figure 5.4 shows the procedure for generating a DES key and an initial vector(IV).

The client module generates a name object from the information that a user gave when he applied a certificate. The attributes for a user to write down are shown in Figure 5.2: name, country, city, organization, telephone number, password, and e-mail. Note that the information which a user gives including the password is not sent to the CA server over the network, but it is sent to the client module. The name object is used in the certification request and X.509 certificate to identify a particular entity, e.g., a user or a computer.

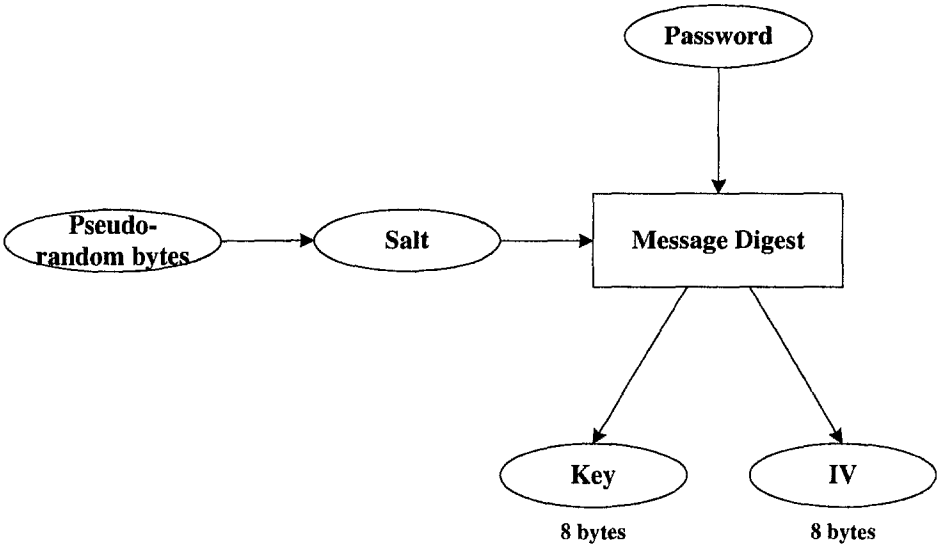


Figure 5.4: DES key and IV generation using password-based encryption algorithm.

The client module also generates a certification request, which will be transformed into a X.509 certificate by the CA server. The certification request consists of three parts: certification request information, a signature algorithm identifier, and a signature on the certification request information.

The certification request information includes a name object and public-key of the user, and is signed with the user's private-key. Note that the signature on the certificate request prevents an opponent from requesting a certificate with another user's public-key.

After finishing the generation of a certificate request, the client module sends the certificate request to the CA server over the network. For this implementation, we generate a socket between the CA server and the client module.

The client module is programmed using the Microsoft Visual C++ 6.0 compiler on the Microsoft Windows 98. The client module generally exists in the CA server and is downloaded when a user accesses the CA server. For this implementation, Microsoft ActiveX control programming technique is used and Visual Basic Script language is also used as a client script language.

5.3 Implementation of Single Sign-On

In this section, the proposed SSO system has been implemented according to the system requirements presented in Section 4.3.3. The implementation of the three components forming the proposed system are respectively described: application server, authentication server, and client.

For the illustration of the behavior of the system, a State Transition Diagram(STD)

will be presented. MSCs are used to view the exchange of messages between system components called instances, while a STD generally focuses on the input and output messages on one instance. As was shown in Section 4.3.3, there exist four types of instances in the system: client, application server, authentication server, and LDAP server. Among them, the STDs of two important instances–application server and authentication server–will be shown.

5.3.1 Implementation of Application Server

An application server gives the first contact point to clients, and provides the authenticated users with various WWW services.

Figure 5.5 shows the STD of an application server, which consists of 2 states and 6 edges; Idle state and Wait state. An Idle state represents an initial state, in which the application server waits for request messages from clients. In the other state, Wait, the application server nominally waits for response messages from the authentication server.

The overview of operations of an application server for its implementation is as follows: First, an application server is in Idle state waiting for a service_req message from clients. If it arrives, the application server checks the URI parameter, which would be expected to include the ticket. This checking process prevents an opponent from directly accessing an application server without authentication process, and is well described in the scenario 10 of Section 4.3.3. If it is a valid request, the application server sets the timer(T_{App}) and changes to Wait state after sending an auth_req message to the authentication server; Otherwise, the result is illustrated in Figure 5.5 as an edge changing from Idle state to Idle state according to a service_req input.

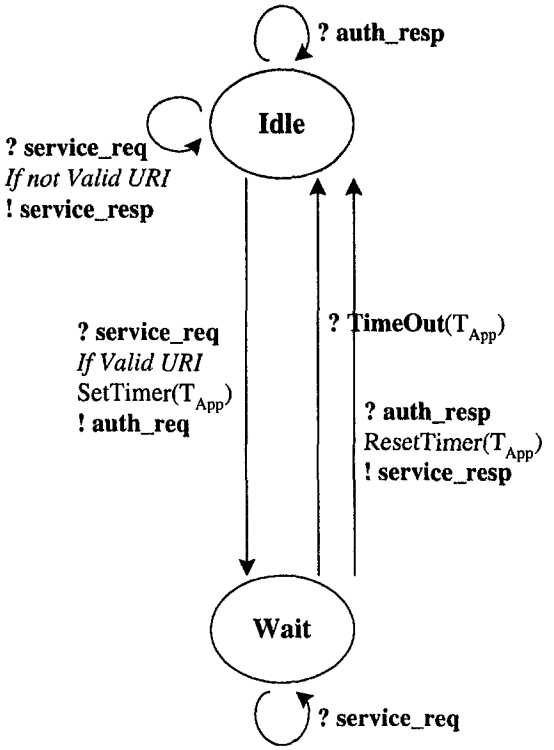


Figure 5.5: State transition diagram of an application server.

If the application server receives an auth_resp message from the authentication server in Wait state, then it checks the ticket parameter. Note that the ticket and the randStr are employed to prevent an opponent from directly accessing an application server without performing authentication protocol. We know that the URI includes the parameters as well as IP address. The application server decrypts the ticket with the secret-key previously shared with the authentication server. If the decryption is successful, the state changes to Idle state. We use the DES-CBC algorithm to generate a ticket and 64 bit all 0's bitstream as an IV.

Another case for transition from Wait state to Idle state is when the timer expires. Think about the case in which a client repeatedly sends service_req messages. The value

of randStr is updated whenever a service_req message arrives at the application server. Therefore, the ticket of authenticated auth_resp message to the previous service_req message would not be considered as valid, if the value of randStr is updated while the auth_resp message returns. In the proposed scheme, a service_req message arriving within timer expiration will be ignored. This is illustrated as an edge changing from Wait state to Wait state according to a service_req input. We set the period of validity of the timer as 30 seconds, which is the same as maximum segment lifetime(MSL). It is the maximum amount of time any TCP segment can exist in the network before being discarded.

It is shown that the application server works well corresponding to the requirements proposed Section 4.3.3. The application server is programmed on Linux platform using CGI technique. To forward message to the authentication server instead of getting back to the client, we use the redirection function using the location header of HTTP.

5.3.2 Implementation of Authentication Server

An authentication server provides other application servers with authentication services and plays an important role in the proposed SSO system.

Figure 5.6 shows the STD of an authentication server. It consists of more states and edges than an application server does; 3 states and 10 edges. The states are Idle, Wait, and Authenticated. The Idle and Wait states are analogous to those of the application server. An Idle state represents an initial state, in which the authentication server waits for request message from an application server. In another state, Wait, the authentication server waits for response messages in order to proceed the authentication process. A state Authenticated means that a client is being authenticated. Trivial edges from one

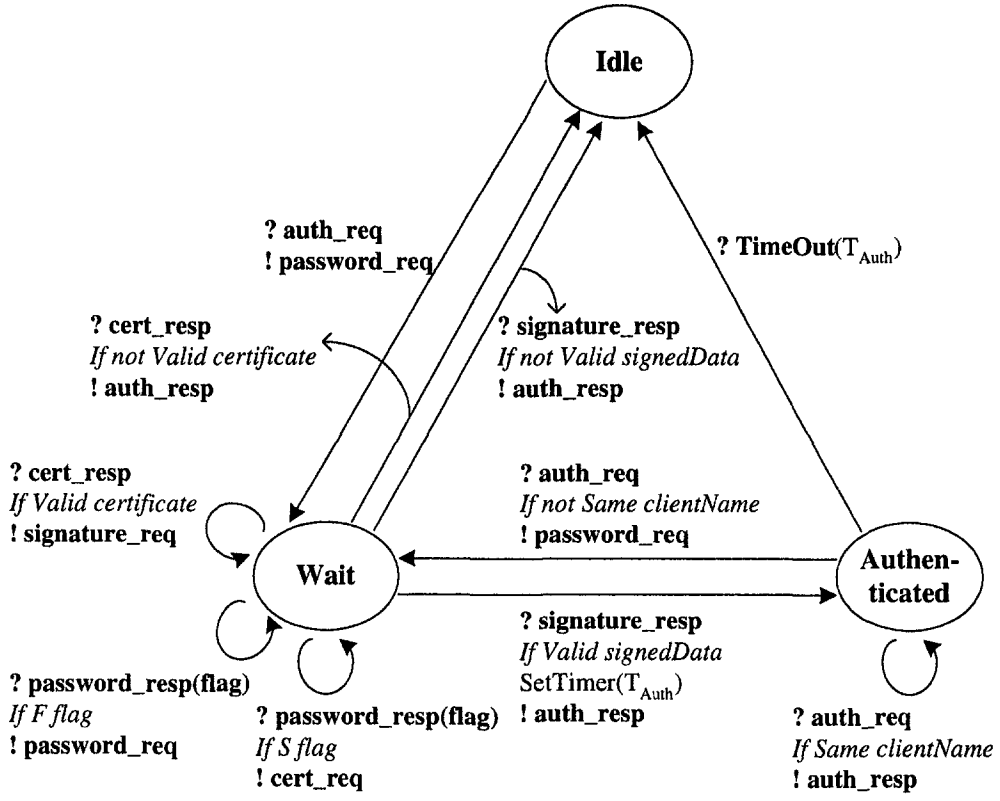


Figure 5.6: State transition diagram of an authentication server.

state to its state are left out in the diagram.

The overview of operations of an authentication server for its implementation is as follows: First, an authentication server is in Idle state waiting for an auth_req message from an application server. If it arrives, the authentication server outputs a password_req message to the client which tried to access the application server, and changes its state into Wait state.

In Wait state, transition to any state is possible. Transition from Wait state to Idle state means that the authentication process fails; Either a cert_resp message with invalid certificate or a signature_resp with invalid signedData arrives.

Transition from Wait state to Wait state means the authentication process's continuous progress. The authentication server stays in Wait state until a signature_resp message with valid signedData arrives. Then the authentication server stores in the local database the authenticated user's userEmail, the network address of the client, and the time that authentication protocol successfully ends. MySQL database management system is used for local database system.

Control of the timer(T_{Auth}) is implemented by checking the log-in time in the database. The period of validity of the timer is set to one hour. Therefore, after one hour, even authenticated user should enter again his password for the authentication process, if he is to access the application server or another application server.

Note that if the authentication server in Authenticated state receives an auth_req message, then if the network address of the client requesting service is the same as that of the previous authenticated client, it still remains in Authenticated state. But if the request comes from another client, transition to Wait state comes about. Timer(T_{Auth}) expiration is another factor to bringing about transition from Authenticated state to Idle state.

The authentication server generates a ticket by encrypting given randStr, which is the parameter of an auth_req message, with the secret-key which have been shared with the application server. The used algorithm is also DES encryption in CBC mode. In addition, the authentication server sends the ticket to the application server as a parameter of an auth_resp message. The format in which the ticket is sent must be in ASCII string to use CGI technique, but the ticket is in encrypted data format, so is binary string. For this implementation, a base-64 encoding/decoding algorithm is also used. Some resulting characters, such as +, /, and =, have the special meaning in the CGI, then they should

be additionally encoded to be sent as CGI parameters.

The authentication server is also programmed on Linux platform using CGI technique like the application server.

5.3.3 Implementation of Client

A client performs intermediate tasks between a user and an application server or an authentication server:

- decrypts the private-key;
- generates the digital signature.

When the client receives a password_req message from the authentication server, then it decrypts the private-key, which has been saved encrypted in the local machine, with the user-given password. The used algorithm is the same as that in the PKI implementation described in Section 5.2.3. If the password is valid, the client sends to the authentication server a password_resp message with S flag.

Then the client generates the digital signature of the randNum, which is the parameter of a signature_req message, using the decrypted private-key. This digital signature is named signedData. We use the RSA signature algorithm with MD5 hash function for the digital signature. The size of modulus is determined as 1024 bit, which is also the same as that used in the PKI implementation.

The client module is programmed using the Microsoft Visual C++ 6.0 compiler on the Microsoft Windows 98, like that in the PKI implementation. The client module exists on the authentication server and is downloaded when the application server requests authentication by sending an auth_req message. For this implementation, Microsoft Ac-

tiveX control programming technique is used and Visual Basic Script language is also used as a client script language.

5.3.4 Verification and Validation of the Proposed System

In this section, the proposed SSO system has been verified and validated using the Specification and Description Language(SDL). Verification generally refers to a consistency check between two different development phases in a development life cycle, whereas validation is the check against the users' needs [58]. SDL is a standard language for specifying and describing systems [59], which has been developed and standardized by ITU-T in the recommendation Z.100.

We describe and specify the proposed system with the SDL, and verify the MSCs and operations of the system. The proposed system has also been validated using the bit-state exploration and random walk method. Validation results show that the proposed SSO system has no deadlock and implicit signal consumption. For the SDL design and simulation, a Telelogic Tau 3.6 SDL suite is used. The system diagram, block diagrams, and process diagrams for SDL description are illustrated in Appendix A.

Chapter 6

Conclusion

User authentication is becoming more and more important issue in security area. So far, most discussions on the information security have been focused on encryption algorithms, but the authentication is now considered very important issue, particularly in the electronic commerce.

Password-based authentication, which has been widely used in most systems, is not suitable for use on current network applications. In addition, it has some security holes to be used as an authentication scheme in the environment where a variety of services are provided from multiple servers. Therefore, cryptographic solution to the problems of both password-based authentication and multi-server authentication has been researched, which leads to the SSO system. A perfect SSO system does not exist yet. The solutions from vendors currently lack all the necessary features, for them to succeed in all environments.

In this dissertation, a SSO system has been designed and implemented. The proposed SSO system satisfies the basic requirement of SSO that a user logs in once using a single password, and gets authenticated access to all servers that the user is authorized to use, without sending the password over the network. The proposed SSO involves the use of digital signature and cryptographic authentication protocol. For the system

to work, a PKI system should be implemented in advance. The proposed PKI includes a CA server, a LDAP directory server, and multiple clients. Its main role is to provide a certificate management infrastructure which issues and supports access to public-key certificates.

In the proposed SSO, we employ the mechanism that a user directly connects the server he tries to access and then the application server automatically requests an authentication service from central server, unlike the existing systems such as Kerberos or Netscape SuiteSpot server in which a user must access a central server to get the evidence that he is authorized.

Implementation results show that the proposed SSO can be easily implemented into the application servers on the Intranet, whose services are based on WWW servers. We also know that the proposed SSO is implemented entirely on the application layer, independent of the transport layer protocols.

The smart card will probably be an integral part of future SSO systems. It may be considered very useful in environments, where smart card readers come into wide use. The smart cards provide a safe storage for a user's private-key, which can be used to authenticate himself securely. A user only bear a simpler PIN in mind to authenticate himself to the smart card, instead of keeping the complicated password with which to securely save the private-key into the local machine. In addition, the authentication can happen with the same mechanism from any properly equipped client with a smart card reader.

References

[1] C. Särs, "Unified single sign-on," in *Proceedings of Helsinki University of Technology*. 1998, Seminar on Network Security: Authorization and Access Control in Open Network Environment, p. Chapter 6, Helsinki University of Technology.

[2] L. Huovinen, "Modern encryption methods in user authentication," in *Proceedings of Helsinki University of Technology*. 1997, Seminar on Network Security: Security of Corporate Networks, p. Chapter 4, Helsinki University of Technology.

[3] H. P. Königs, "Cryptographic identification methods for smart cards in the process of standardization," *IEEE Communications Magazine*, vol. 29, no. 6, pp. 42–48, June 1991.

[4] J. Hursti, "Single sign-on," in *Proceedings of Helsinki University of Technology*. 1997, Seminar on Network Security: Security of Corporate Networks, p. Chapter 2, Helsinki University of Technology.

[5] T. Tervo, "Single sign-on solutions in a mixed computing environment," in *Proceedings of Helsinki University of Technology*. 1998, Seminar on Network Security: Authorization and Access Control in Open Network Environment, p. Chapter 7, Helsinki University of Technology.

[6] G. A. Champine, D. E. Geer, and W. N. Ruh, "Project Athena as a distributed computer system," *IEEE Computer*, vol. 23, no. 9, pp. 40–51, Sept. 1990.

[7] B. C. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 33–38, Sept. 1994.

[8] S. M. Bellovin and M. Merritt, "Limitations of the kerberos authentication system," *Computer Communications Review*, vol. 20, no. 5, pp. 119–132, Oct. 1990.

[9] M. A. Sirbu and J. C. .Chuang, "Distributed authentication in Kerberos using public key cryptography," in *Network and Distributed System Security*, Feb. 1997, pp. 134–141.

[10] S. Chokhani, "Toward a national public key infrastructure," *IEEE Communications Magazine*, vol. 32, no. 9, pp. 70–74, Sept. 1994.

[11] M. Branchaud, "A survey of public-key infrastructure," M.S. thesis, Dept. of Computer Science, McGill University, Montreal, 1997.

[12] Netscape Communications Corporation, *Single Sign-On Deployment Guide: Security*, Netscape Communications Corporation, 1997.

[13] H. Feistel, "Cryptography and computer privacy," *Scientific American*, vol. 228, no. 5, pp. 15–23, May 1973.

[14] C. E. Shannon, "Communication theory of secrecy systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[15] National Bureau of Standards, *Data Encryption Standard*, NBS FIPS PUB 46, Jan. 1977.

[16] National Bureau of Standards, *Guidelines for Implementing and Using the NBS Data Encryption Standard*, NBS FIPS PUB 74, Apr. 1981.

[17] National Bureau of Standards, *DES Modes of Operations*, NBS FIPS PUB 81, Dec. 1980.

[18] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Upper Saddle River, NJ: Prentice-Hall, 2nd edition, 1999.

[19] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C*, John Wiley & Sons, second edition, 1996.

[20] E. Roback and M. Dworkin, "First advanced encryption standard(aes) candidate conference," *Cryptobytes*, vol. 4, no. 2, pp. 6–14, Win. 1999.

[21] RSA Laboratories, *Frequently Asked Questions About Today's Cryptography: V4.0*, RSA Laboratories, 1998.

[22] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE trans. Information Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.

[23] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signature and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[24] N. Koblitz, *A Course in Number Theory and Cryptography*, New York: Springer-Verlag, 1987.

[25] National Institute of Standards and Technology, *Digtal Signature Standard*, NIST FIPS PUB 186, May 1994.

[26] T. ElGamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," *IEEE trans. Information Theory*, vol. IT-31, no. 4, pp. 469–472, July 1985.

[27] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press, 1995.

[28] A. Fiat and A. Shamir, "How to prove yourself: practical solution to identification and signature problems," in *Advances in Cryptology - Crypto '86*. 1987, vol. 263 of *Lecture Notes in Computer Science*, pp. 186–194, Springer-Verlag.

[29] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," in *17th ACM Symposium on Theory of Computing*, May 1985, pp. 291–304.

[30] L. C. Guillou and J. J. Quisquater, "A practical zero-knowledge protocol fitted to security microprocessor minimizing both transmission and memory," in *Advances in Cryptology - Eurocrypt '88*. 1988, vol. 330 of *Lecture Notes in Computer Science*, pp. 123–128, Springer-Verlag.

[31] C. P. Schnorr, "Efficient signature generation for smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, Mar. 1991.

[32] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Advances in Cryptology - Crypto '84*. 1985, vol. 196 of *Lecture Notes in Computer Science*, pp. 47–53, Springer-Verlag.

[33] P. Wing and B. O'Higgins, "Using public-key infrastructures for security and risk management," *IEEE Communications Magazine*, vol. 37, no. 9, pp. 71–73, Sept. 1999.

[34] W. E. Burr, *Public Key Infrastructure Technical Specifications(Version 1): Part C–Concept of Operations*, TWG-95-108, Feb. 1996.

[35] CCITT, *The Directory: Authentication Framework*, Recommendation X.509, 1989.

[36] CCITT, *The Directory: Overview of Concepts, Models, and Service*, Recommendation X.500, 1988.

[37] W. Yeong, T. A. Howes, and S. Kille, *Lightweight Directory Access Protocol*, RFC 1777, Mar. 1995.

[38] T. A. Howes, S. Kille, W. Yeong, and C. Robbins, *The String Representation of Standard Attribute Syntaxes*, RFC 1778, Mar. 1995.

[39] T. A. Howes, "The lightweight directory access protocol: X.500 lite," Tech. Rep. CITI 95-8, Center for Information Technology Integration, University of Michigan, July 1995.

[40] T. A. Howes and M. C. Smith, "A scalable, deployable directory service framework for the internet," Tech. Rep. CITI 95-7, Center for Information Technology Integration, University of Michigan, July 1995.

[41] B. S. Kaliski Jr., *An Overview of the PKCS Standards*, RSA Laboratories, Nov. 1993.

[42] RSA Laboratories, *PKCS #1: RSA Cryptography Standard*, RSA Laboratories, Sept. 1998, Version 2.0.

[43] R. L. Rivest, *The MD5 Message Digest Algorithm*, RFC 1321, Apr. 1992.

[44] RSA Laboratories, *PKCS #5: Password-Based Cryptography Standard*, RSA Laboratories, Mar. 1999, Version 2.0.

[45] R. L. Rivest, *The MD4 Message Digest Algorithm*, RFC 1320, Apr. 1992.

[46] RSA Laboratories, *PKCS #7: Cryptographic Message Syntax Standard*, RSA Laboratories, Nov. 1993, Version 1.5.

[47] J. Linn, *Privacy Enhancement for Internet Electronic Mail: Part 1- Message Encipherment and Authentication Procedures*, RFC 1421, Feb. 1993.

[48] RSA Laboratories, *PKCS #8: Private-Key Information Syntax Standard*, RSA Laboratories, Nov. 1993, Version 1.2.

[49] RSA Laboratories, *PKCS #10: Certification Request Syntax Standard*, RSA Laboratories, May 2000, Version 1.7.

[50] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, Dec. 1978.

[51] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer, "Section e.2.1: Kerberos Authentication and Authorization System," Tech. Rep., MIT Project Athena, Dec. 1987.

[52] J. G. Steiner, B. C. Neuman, and J. I. Schiller, "Kerberos: an authentication service for open network systems," in *USENIX Conference Proceedings*, Feb. 1988, pp. 191–202.

[53] J. T. Kohl, B. C. Neuman, and T. Ts'o, "The evolution of the Kerberos authentication system," *Distributed Open Systems*, pp. 78–94, 1994.

[54] E. Rudolph, P. Graubmann, and J. Grabowski, "Tutorial on message sequence charts," *Computer Networks and ISDN Systems*, vol. 28, no. 12, pp. 1629–1641, Dec. 1996.

[55] R. Weltman and T. Dahbura, *LDAP Programming with Java*, Addison-Wesley, 2000.

[56] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[57] National Institute of Standards and Technology, *Secure Hash Standard*, NIST FIPS PUB 180-1, Apr. 1995.

[58] D. Hogrefe, "Validation of SDL systems," *Computer Networks and ISDN Systems*, vol. 28, no. 12, pp. 1659–1667, Dec. 1996.

[59] R. Braek, "SDL basics," *Computer Networks and ISDN Systems*, vol. 28, no. 12, pp. 1585–1602, Dec. 1996.

Appendix A

System Description in SDL

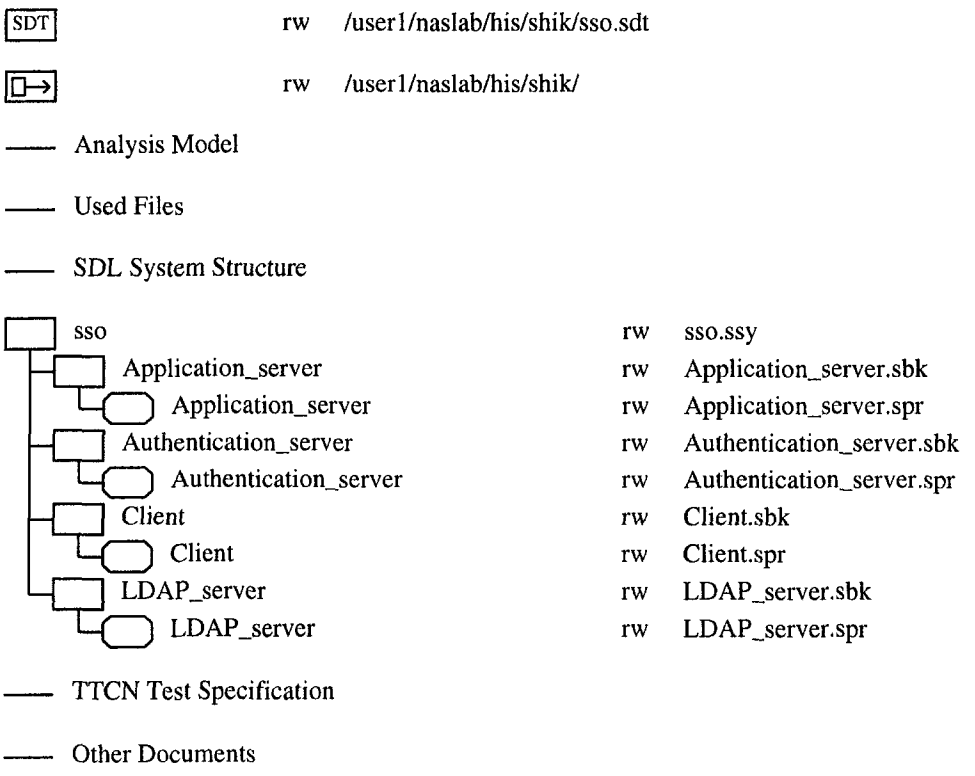


Figure A.1: Organizer View of the proposed SSO system.

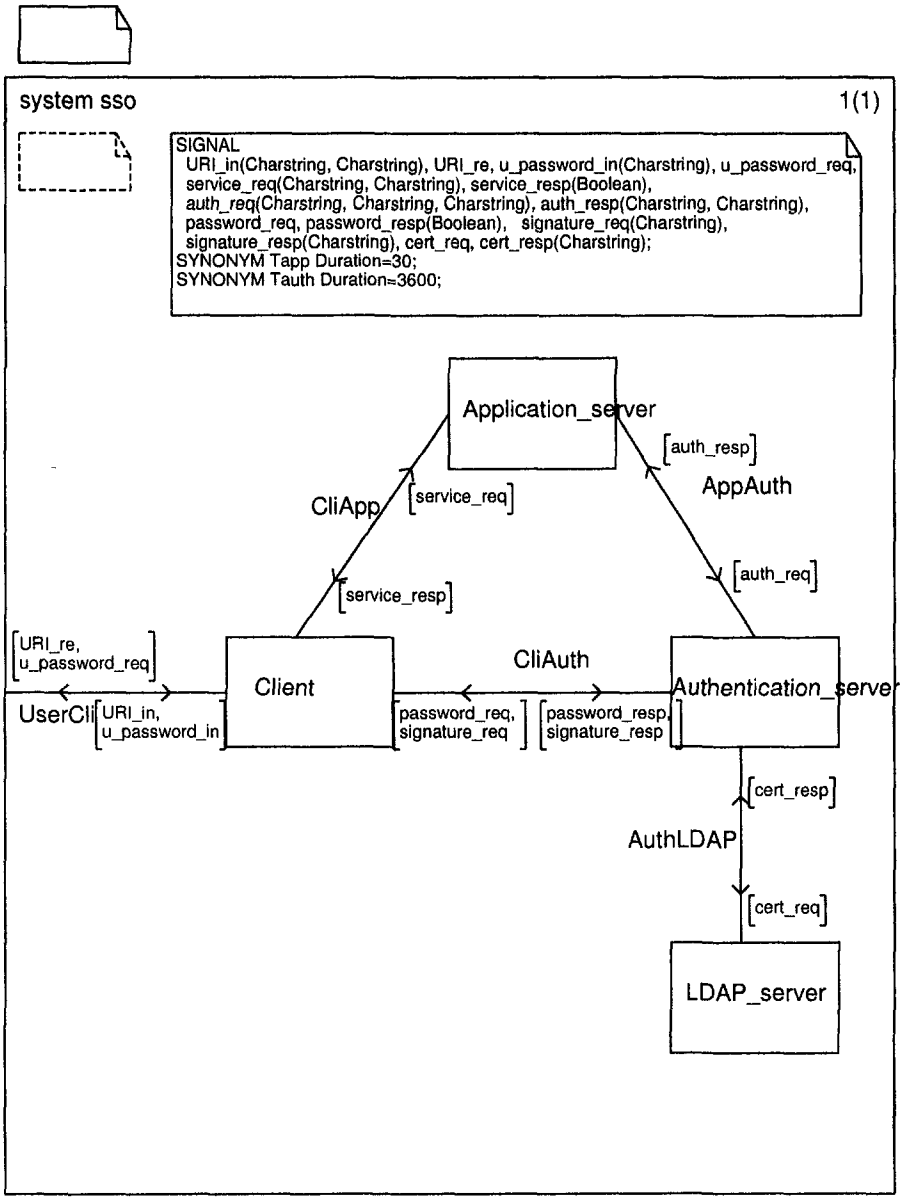


Figure A.2: System diagram of the proposed SSO system.

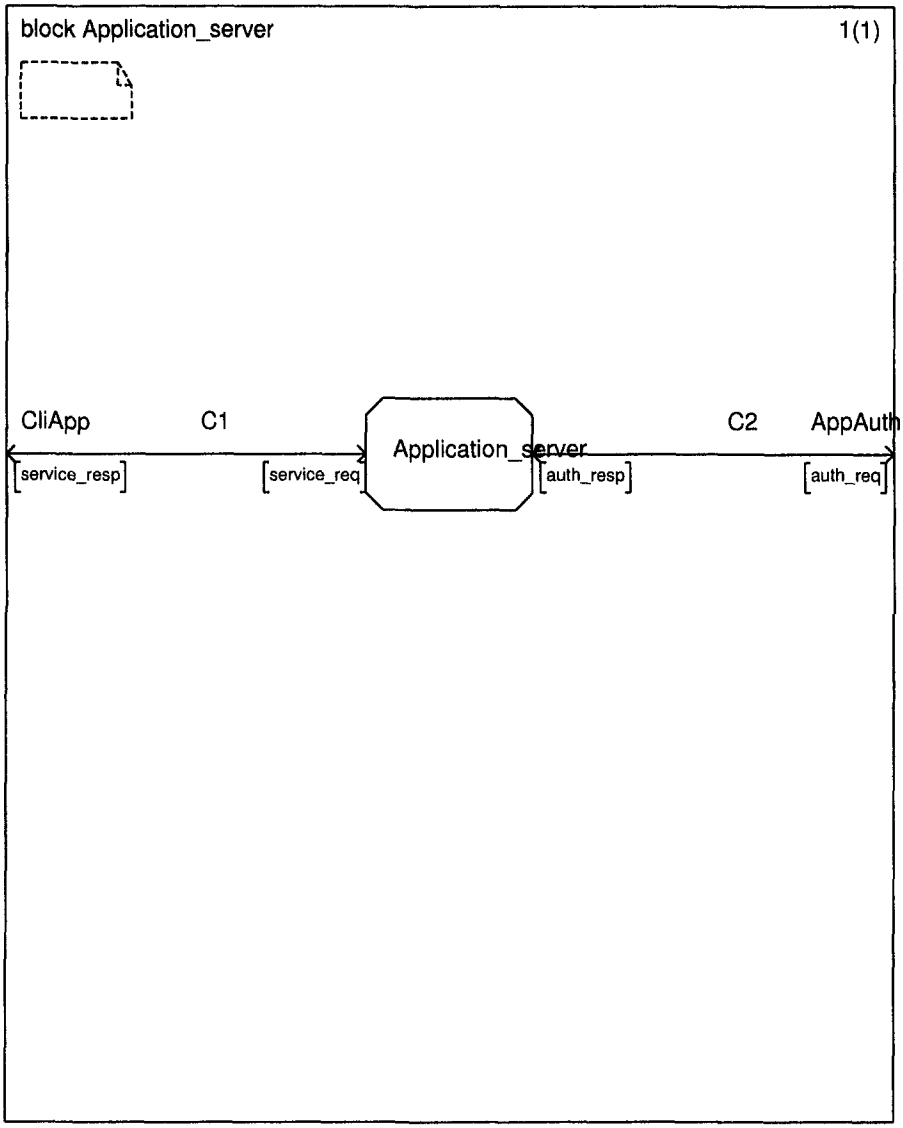


Figure A.3: Block diagram of the application server.

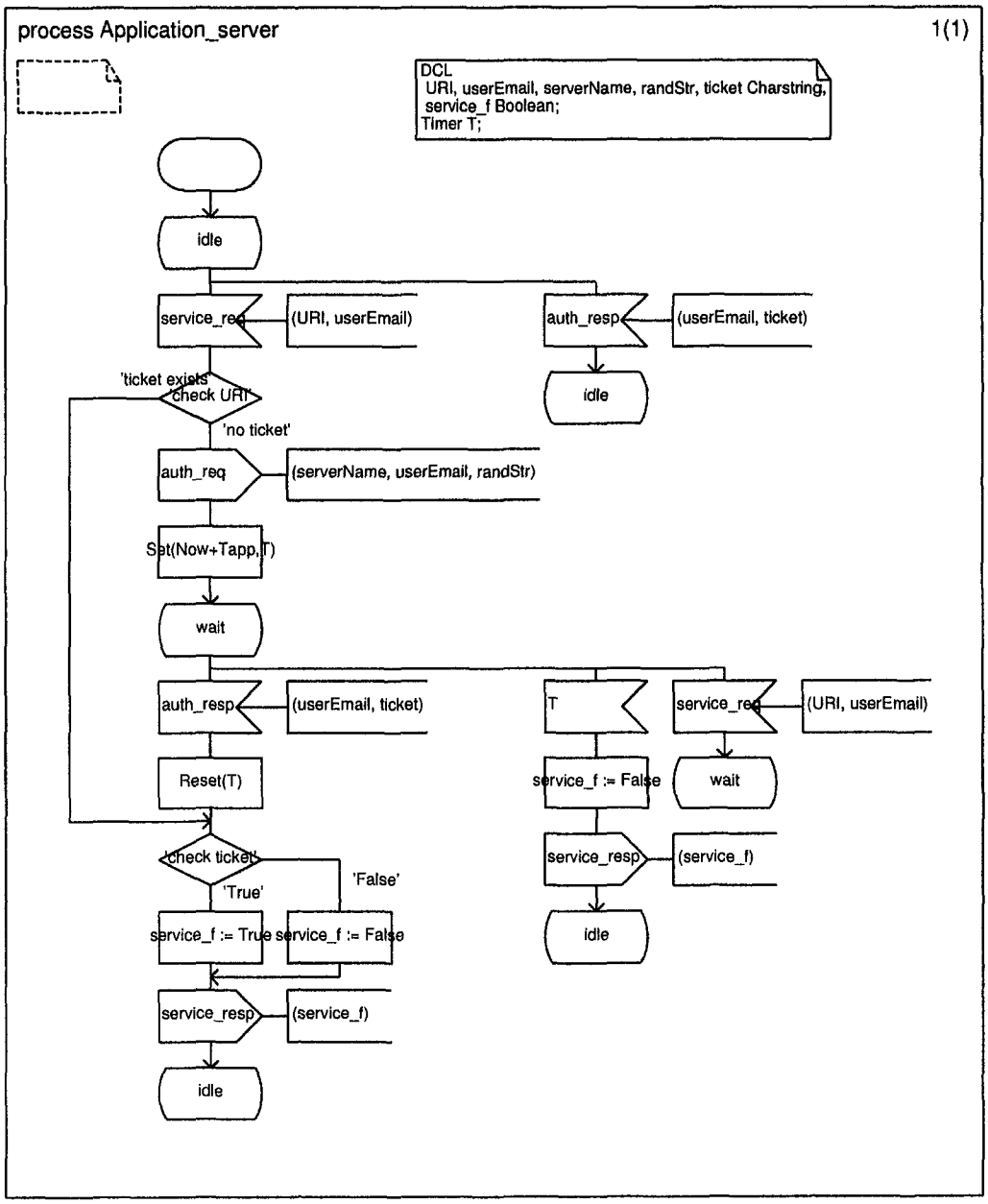


Figure A.4: Process diagram of the application server.

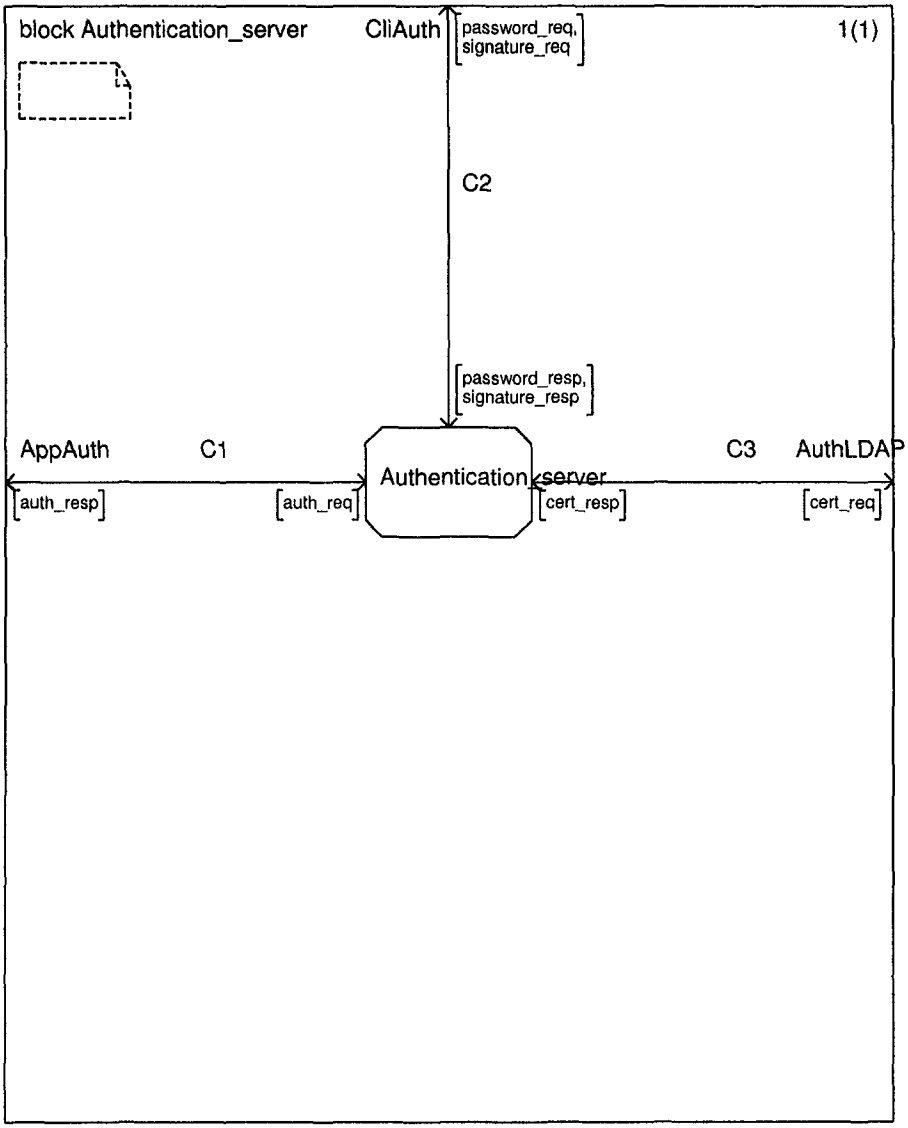


Figure A.5: Block diagram of the authentication server.

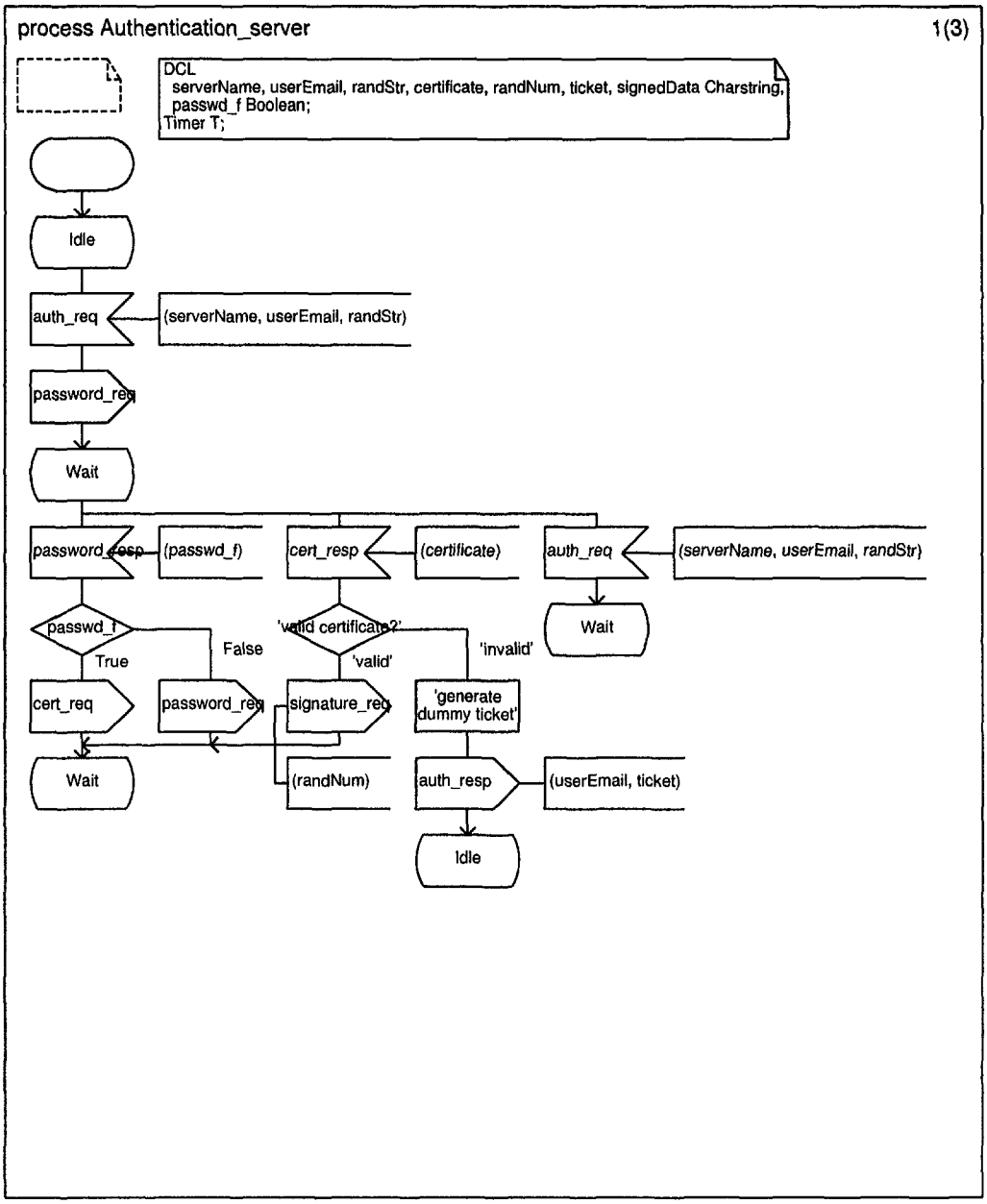


Figure A.6: Process diagram of the authentication server.

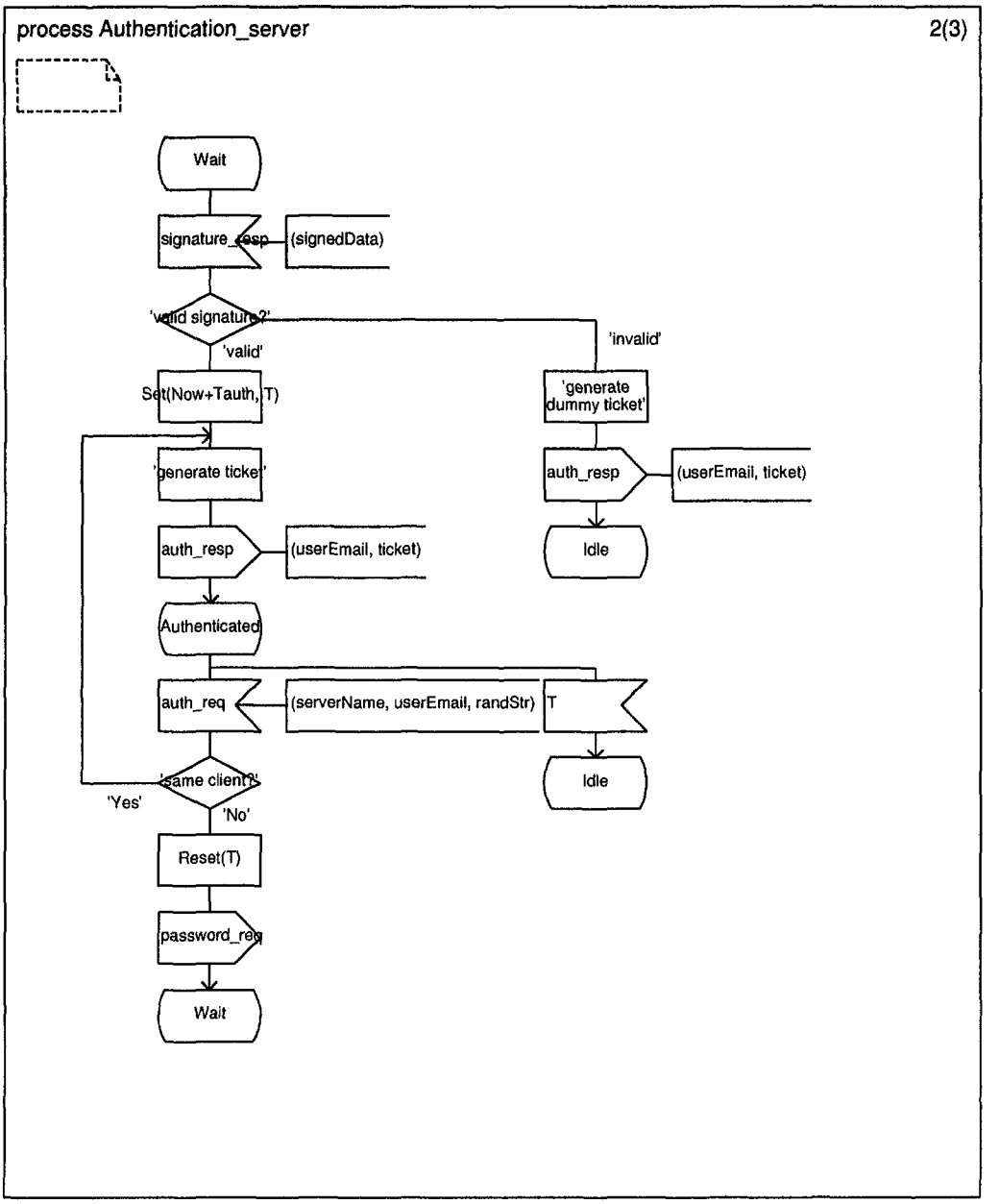


Figure A.7: Process diagram of the authentication server: *continued.*

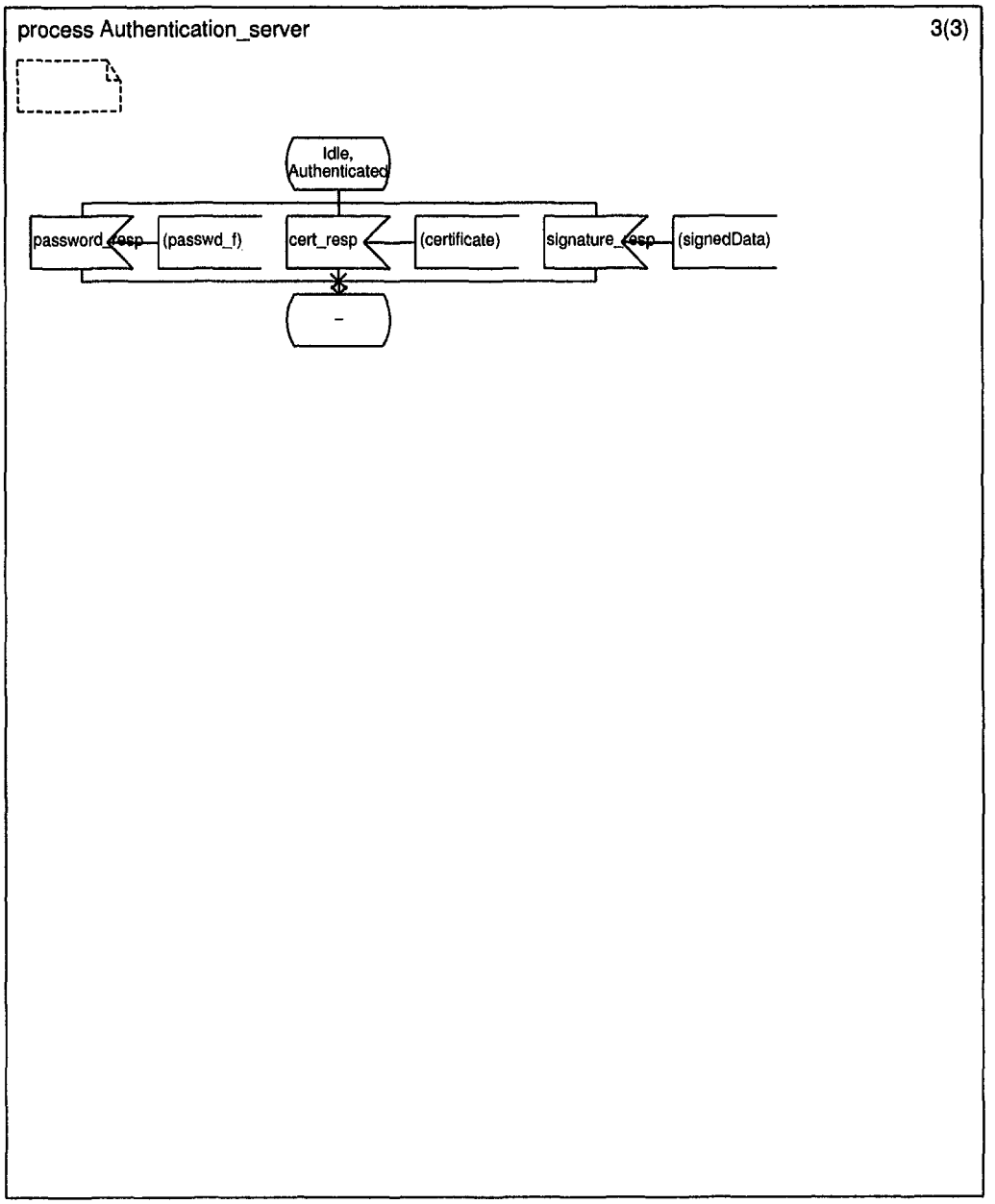


Figure A.8: Process diagram of the authentication server: *continued.*

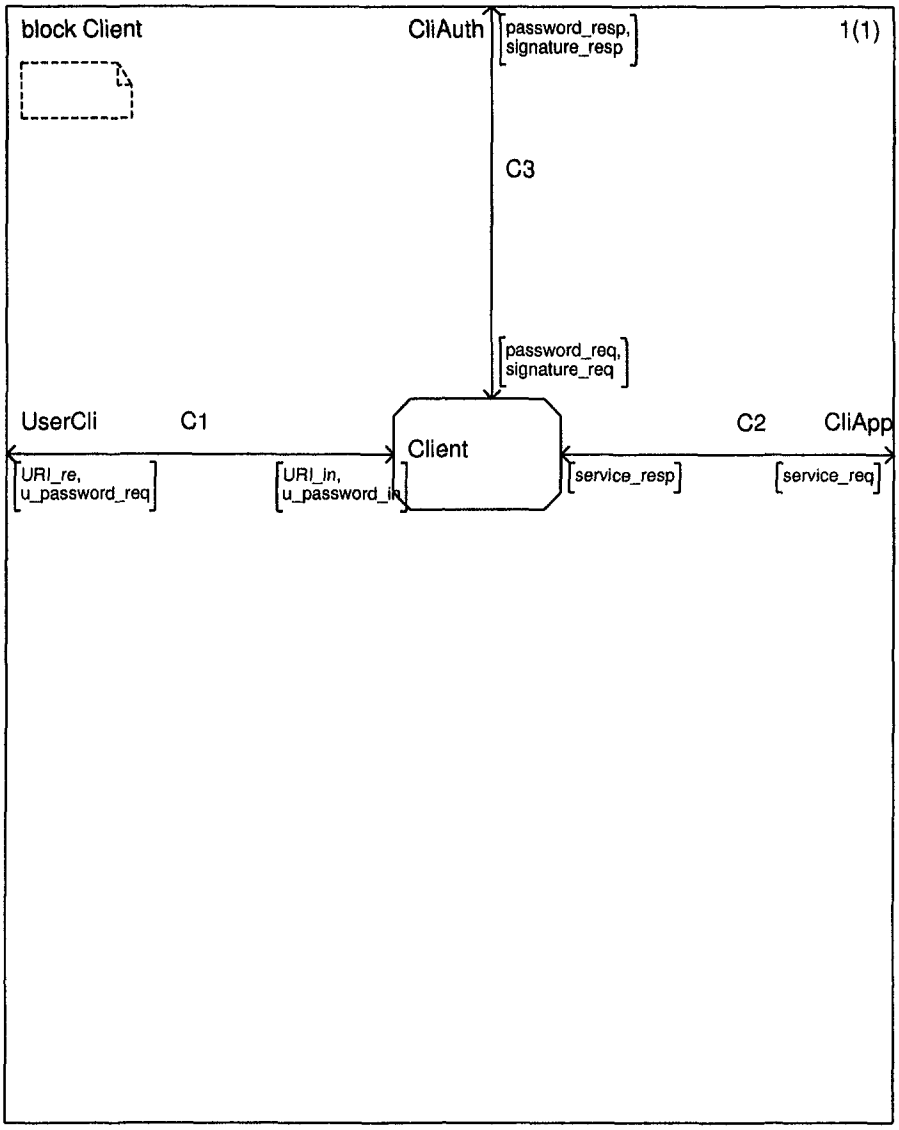


Figure A.9: Block diagram of the client.

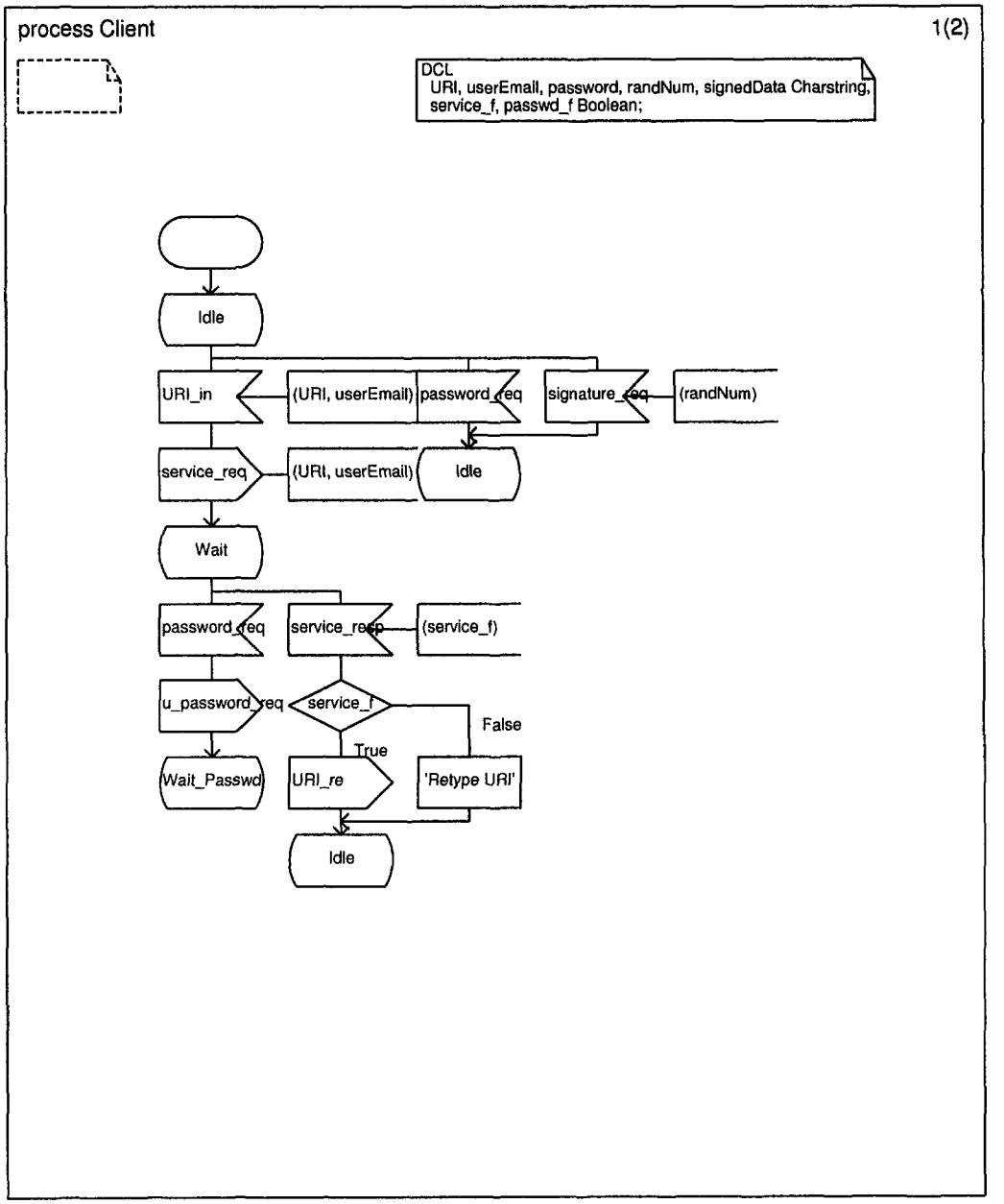


Figure A.10: Process diagram of the client.

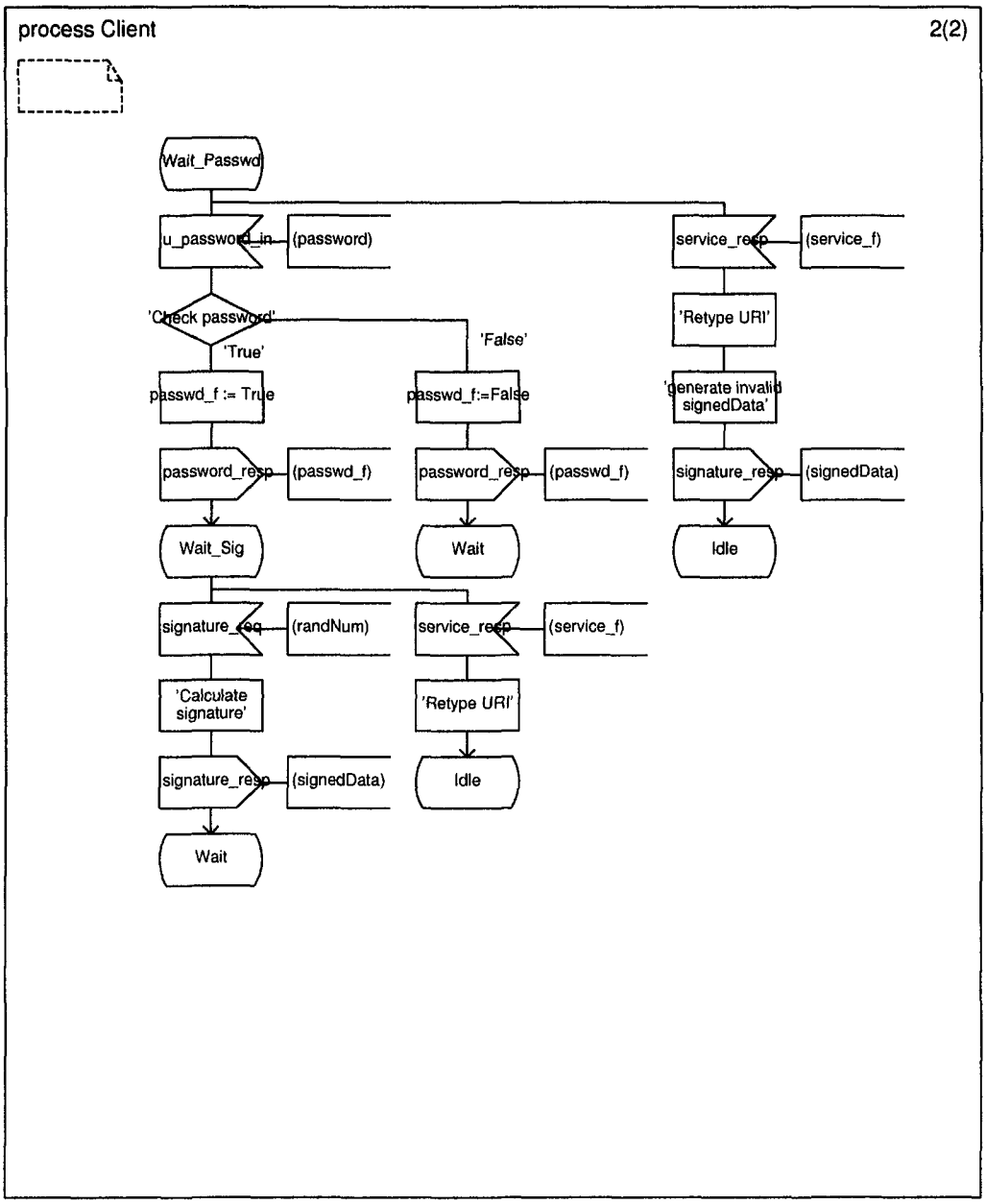


Figure A.11: Process diagram of the client: *continued.*

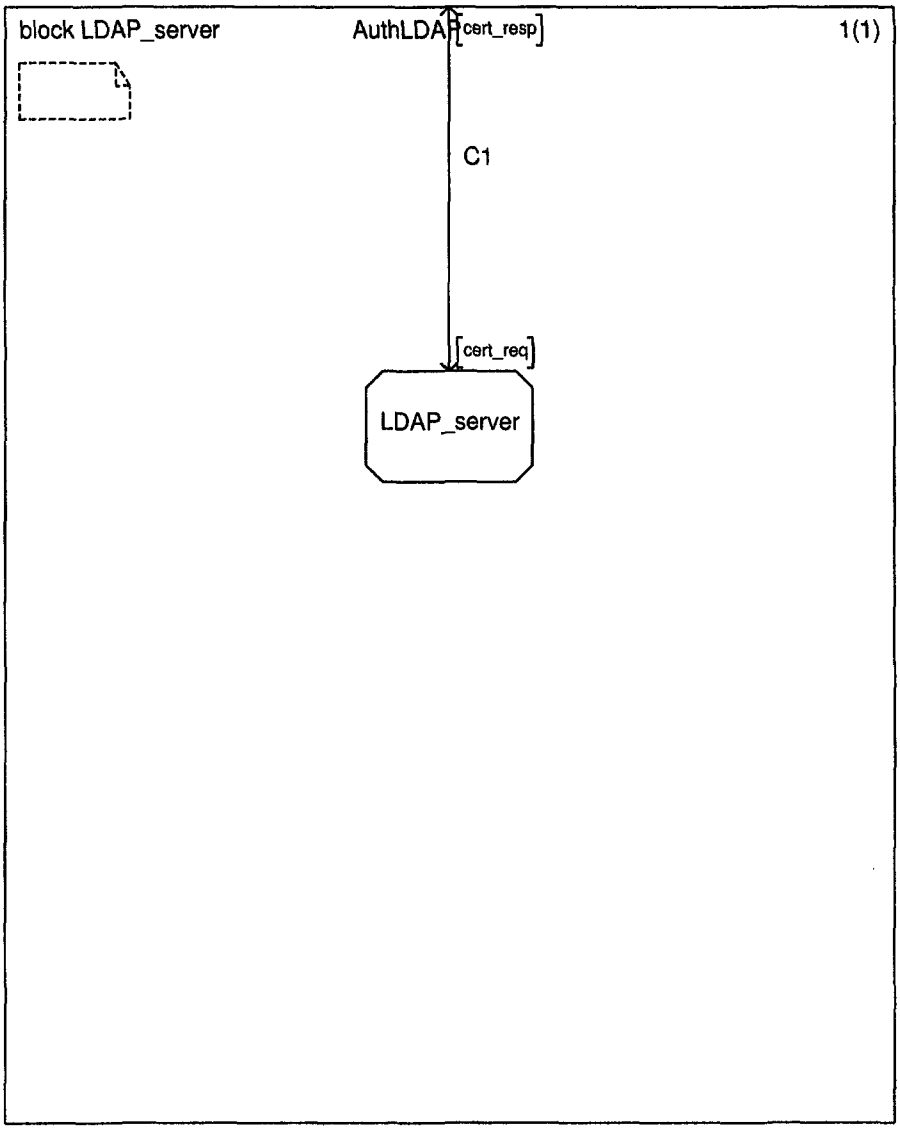


Figure A.12: Block diagram of the LDAP server.

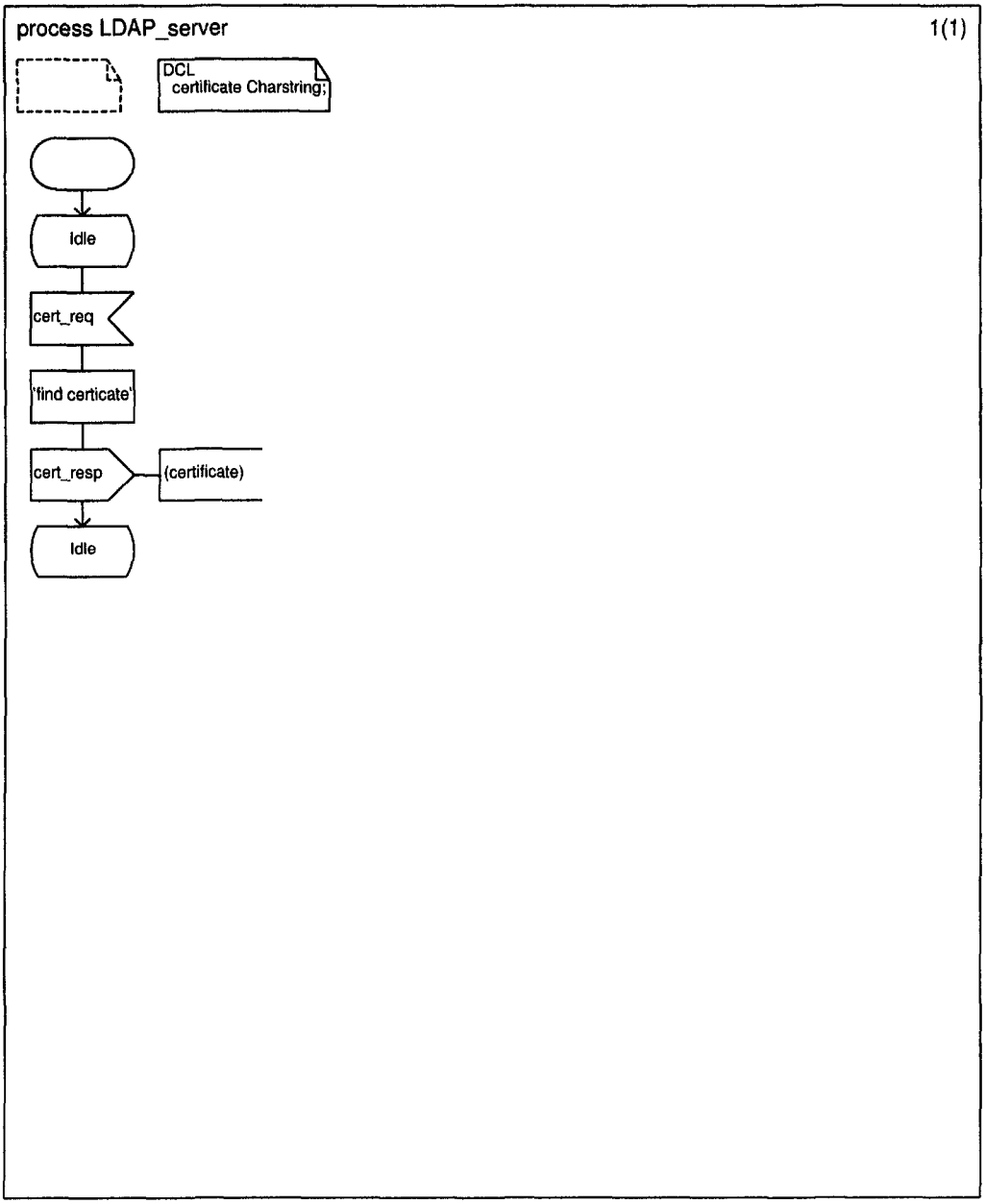


Figure A.13: Process diagram of the LDAP server.

국문요약

공개키 기반 구조의 구축을 통한 인트라넷 통합 인증 시스템 설계 및 구현

최근 정보통신 및 인터넷의 발달로 네트워크를 통한 서비스 제공의 기회가 확대됨에 따라 전송되는 정보에 대한 보호와 정당한 사용자에 대한 인증 문제가 중요한 연구 분야로 대두되고 있다. 현재 사용자를 인증하기 위한 방법으로는 패스워드 방식이 널리 사용되고 있지만, 네트워크 상에서 패스워드가 재사용될 수 있다는 문제점 때문에 안전한 인증 방식으로 사용되기에는 문제점이 있다. 이러한 문제점을 해결하기 위해 암호이론에 기반한 challenge-response 프로토콜을 이용한 인증 방식이 제안되었다.

그러나, 이러한 인증 방식들은 다중 서버가 운영되는 인트라넷 환경에 사용되기에는 부적절하다. 즉 각 서버 당 다른 패스워드를 사용해야 하는 요구사항은 사용자 및 관리자 모두에게 부담으로 작용한다. 또한 challenge-response 프로토콜이 사용된다 하더라도 역시 다중 서버 환경에서는 구현 및 관리가 용이하지 않다.

이러한 구현상의 여러 단점들을 극복하기 위해 통합 인증 시스템(SSO)의 개념이 도입되었다. 통합 인증 시스템에서는 사용자가 하나의 패스워드를 이용해서 인트라넷 환경하의 모든 응용 서버에 접속할 수 있다. 또한 사용자가 입력한 패스워드가 네트워크를 통해서 전송되지 않으므로 패스워드 재사용 공격으로부터 안전할 수 있다.

본 논문에서는 전자서명과 암호학적 challenge-response 프로토콜에 기반한

통합 인증 시스템이 설계 및 구현되었다. 공개키 암호 시스템의 사용을 위해 공개키 기반 구조 역시 설계 및 구현되었다. 제안한 공개키 기반 구조는 인증서 서버, 디렉토리 서버, 클라이언트 모듈로 구성되어 있으며, 통합 인증 시스템의 challenge-response 프로토콜의 구현을 위해 인트라넷 각 사용자에게 인증서를 생성, 발급해주는 역할을 수행한다.

제안한 통합 인증 시스템은 공개키 기반 구조, 인증 서버, 응용 서버와 클라이언트들로 구성되어 있으며, 웹 서비스가 제공되며 엄격한 인증을 필요로 하는 인트라넷으로의 구현을 목표로 한다. 제안한 시스템은 응용 계층에서 구현되므로 네트워크 및 수송 계층 프로토콜에 독립적이며, 인증 서버 및 응용 서버에 접속 시 클라이언트 모듈을 웹 브라우저에 다운로드함으로써 쉽게 구현, 이용될 수 있다.

핵심되는 말 : 인증, 통합 인증 시스템, 전자 서명, 공개키 기반 구조