SECURE ATTRIBUTE SERVICES ON THE WEB

by

Joon S. Park

A Dissertation

Submitted to the

Faculty of the Graduate School

of

George Mason University

in Partial Fulfillment of

the Requirements for the Degree

of

Doctor of Philosophy

Information Technology

Committee:

_____ Dr. Ravi Sandhu, Dissertation Director

_____ Dr. Edgar Sibley

_____ Dr. Prasanta Bose

_____ Dr. Elizabeth White

_____ Dr. Stephen G. Nash, Associate Dean, Graduate Studies and Research

_____ Dr. Lloyd J. Griffiths, Dean, School of Information Technology and Engineering

Date: _____

Summer 1999

George Mason University

Fairfax, Virginia

SECURE ATTRIBUTE SERVICES ON THE WEB

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University.

JOON S. PARK

Bachelor of Science, Yonsei University, Seoul, Korea, February 1989
Master of Science, George Mason University, Fairfax, Virginia, U.S.A., May 1997

Director: Dr. Ravi Sandhu, Full Professor
Information and Software Engineering

Summer 1999
George Mason University
Fairfax, Virginia

DEDICATION

TO MY GOD

"The fear of the Lord is the beginning of wisdom, and the knowledge of the Holy One is understanding" (Proverbs 9:10).

TO MY PARENTS

They prayed, supported, and made many sacrifices, so that I could obtain an education.

TO MY WIFE AND BROTHER

They loved me and encouraged me to do my best.

ACKNOWLEDGMENTS

I sincerely express my gratitude to my dissertation director, Dr. Ravi Sandhu, for the invaluable guidance and direction he has provided to me during the course of my study and research at GMU. I was fortunate to have met such an outstanding scholar, who is also an understanding and supportive advisor.

I would like to thank my committee members, Dr. Edgar Sibley, Dr. Prasanta Bose, and Dr. Elizabeth White for providing me with their thoughtful comments and guidance.

Special thanks go to my family for their patient support, sacrifices, and encouragement. I always thank my God for blessing me with such a wonderful family.

I appreciate the assistance and input from the following people: SreeLatha Ghanta, Gail J. Ahn, Soonam Kahng, and Eonsuk Shin.

Finally, I acknowledge the financial support by NSA (National Security Agency), NRL (Naval Research Laboratory), NIST (National Institute of Standards and Technology), and NSF (National Science Foundation) for this research.

ABSTRACT

SECURE ATTRIBUTE SERVICES ON THE WEB

Joon S. Park, Ph.D.
George Mason University, 1999
Dissertation Director: Dr. Ravi Sandhu

Increased integration of Web, operating system, and database system technologies will lead to continued reliance on Web technology for enterprise computing. Therefore, a successful marriage of the Web and security technologies has the potential for considerable impact on large-scale systems and their deployment.

The motivation behind this research is to protect attributes from possible attacks on the Web. An attribute is a particular property of an entity, such as a role, access identity, group, or clearance. If attributes are provided integrity, authentication, and confidentiality on the Web, Web servers can then use these secure attributes for many purposes, for instance, access control, authorization, authentication, or electronic transactions.

This dissertation identifies the *user-pull* and *server-pull* models for secure attribute services on the Web. It describes the development of novel enhancements, called *secure cookies* and *smart certificates*, to existing Web technologies to provide improved security services on the Web. To demonstrate concrete examples of secure attribute services and show feasibility of these novel techniques, secure cookies and smart certificates are used (separately) to implement role-based access control on the Web.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

The World Wide Web (WWW) is a critical enabling technology for electronic commerce and business on the Internet. Its underlying protocol, HTTP (HyperText Transfer Protocol [FGM98]), has been widely used to synthesize diverse technologies and components for great effect in Web environments. WWW is commonplace. Increased integration of Web, operating system, and database system technologies will lead to continued reliance on Web technology for enterprise computing. However, current approaches to access control on Web servers are mostly based on individual users; therefore, they do not scale to enterprise-wide systems.

An attribute is a particular property of an entity, such as a role, access identity, group, or clearance. If the attributes of individual users are provided securely on the Web by security services - such as authentication, integrity, and confidentiality - we can use those attributes for many purposes, including attribute-based access control, authorization, authentication, and electronic transactions. Therefore, a successful marriage of the Web and secure attribute services has potential for considerable impact on and deployment of effective enterprise-wide security in large-scale systems.

1.2 Approach

In this research, we present a comprehensive approach to secure attribute services on the Web. We identify the *user-pull* and *server-pull* models and analyze their advantages and disadvantages. To support these models on the Web, we take relatively mature technologies and extend them for secure attribute services on the Web. In order to do so, we make use of third technologies that are being used on the Web: *Cookies* and *X.509*.

First, we investigate how to secure and use the very popular Web technology of cookies [KM97, KM98a, KM98b, Lau98] for secure attribute services on the Web. Cookies were invented to maintain continuity and state on the Web. Cookies contain strings of text characters encoding relevant information about the user, and are sent to the user's hard disk via the browser while the user is visiting a cookie-using Web site. The Web server gets those cookies back and retrieves the user's information from the cookies when the user later returns to the same Web site. The purpose of a cookie is to acquire information and use it in subsequent communications between the Web server and the browser without asking for the same information again. However, it is not safe to store and transmit this sensitive information in cookies because cookies are insecure. Cookies are stored and transmitted in clear text, which is readable and can be easily forged. One contribution of this dissertation is to identify and discuss techniques to make cookies secure to carry and store sensitive data in them.[1] We name these cookies *secure cookies*. These techniques have varying degrees of security and convenience for users and system administrators.

Second, we also use X.509v3 [ITU93, ITU97, HFPS98], an ISO standard, since

[1] Our motivation for using the cookie mechanism is that it is already widely deployed in existing Web browsers and servers for maintaining state on the Web. There are other techniques to make Web transactions secure without the use of secure cookies. For example, the secure HTTP protocol (s-HTTP) and HTML security extensions [RS98, SR98] can be used for this purpose. Other protocols and extensions could be devised to operate in conjunction with the Secure Sockets Layer (SSL) protocol [WS96, DA99]. However, these technologies cannot solve the stateless problem of HTTP. Furthermore, none of these technologies can prevent end-sytem threats (described in Section 4.2) to cookies.

public-key infrastructure (PKI) has been recognized as a crucial enabling technology for security in large-scale networks. The basic purpose of X.509 certificates is simply the binding of users to keys. Even though X.509 has the ability to be extended, the application of the extensions of X.509 for secure attributes has not yet been precisely defined. This dissertation shows how to extend and use existing X.509 certificates for secure attribute services on the Web. We name these extended X.509 certificates *smart certificates*. Smart certificates have several sophisticated features: they are able to support short-lived lifetime and multiple CAs, contain attributes, provide postdated and renewable certificates, and provide confidentiality. Selection of these new features depends on applications.

This dissertation describes several techniques to make secure cookies and smart certificates so as to enable secure attribute services between existing Web servers and browsers, and introduces examples of electronic commerce and business applications for use of these new technologies. To prove the feasibility of these new technologies, we implement RBAC (Role-Based Access Control [San98]) on the Web as one possible application of secure attribute services by using two different methods: secure cookies and smart certificates.

1.3 Organization of the Dissertation

In Chapter 2, we describe the technologies most relevant to our work, such as cookies, SSL (Secure Socket Layer), X.509 (Public-Key Certificate), attribute certificates, and PGP (Pretty Good Privacy). In Chapter 3, we identify operational models for secure attribute services on the Web. Chapter 4 addresses security threats to cookies and describes how to render secure cookies on the Web using cryptographic technologies. Chapter 5 describes how to extend X.509 certificates with new features to provide secure attribute services on the Web. To prove the feasibility of these new technologies, in Chapter 6, we implement RBAC (Role-Based Access Control) on the Web using secure cookies and smart certificates separately. Finally, Chapter 7 gives our conclusions.

CHAPTER 2

RELATED TECHNOLOGIES

2.1 Cookies

Electronic commerce and business on the Internet is facilitated in large part by the World Wide Web. The HyperText Transport Protocol (HTTP [FGM98]) carries all interactions between Web servers and browsers. Since HTTP is stateless, it does not support continuity for browser-server interaction between successive user visits. Without a concept of a session in HTTP, users are strangers to a Web site every time they access a page in a Web server.

Cookies [KM97, KM98a, KM98b, Lau98] were invented to maintain continuity and state on the Web. They contain strings of text characters encoding relevant information about the user, and are sent to the user's computer (RAM) via the browser while the user is visiting a cookie-using Web site. After the browser is closed, usually, the cookies are stored in the user's hard disk. The Web server gets those cookies back and retrieves the user's information from the cookies when the user later returns to the same Web site. The purpose of a cookie is to acquire information and use it in subsequent communications between the server and the browser without asking for the same information again. Often a server will set a cookie to hold a pointer, such as an identification number, as a user-specific primary key of the information database maintained in the server.

At present, there are many browsers that support cookies, including Netscape, MS Internet Explorer, GNNWorks, NetCruiser, and OmniWeb. There are basically

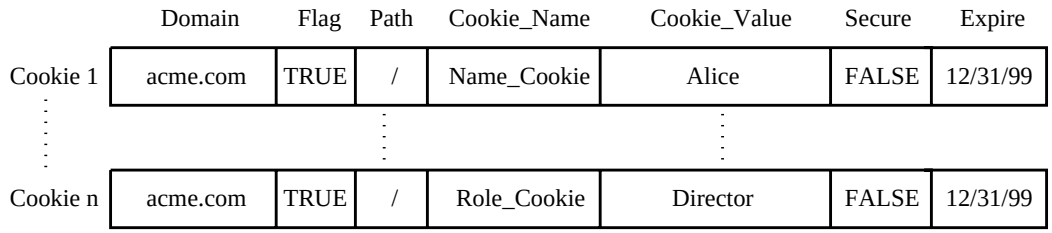


Figure 2.1: An Example of Cookies on the Web

three modes of cookie management in browsers. The first mode is to simply refuse all cookies without asking. The second mode is to accept all cookies, but offer users the ability to delete all of them at once. The last provides more options to users. A user can decide to accept cookies from designated sites.

Cookies have been used for many purposes on the Web, such as selecting display mode (e.g., frames or text only), making shopping cart selections, and carrying names, passwords, account numbers, or some other bits of identifying data on the user's machine. The possible applications are endless, since cookies are a general mechanism to save and replay relevant application-dependent data. Contrary to some discussions in the popular press, cookies cannot interact with other data on the user's hard drive, nor can they capture anything from it. They can only save and retrieve information placed therein by Web servers.

Although there are many ways to use cookies on the Web, the basic process and the contents of cookies are outlined in the following description. Figure 2.1 shows an example of cookies. To create a cookie for a Web site, the Web server sends a *Set_Cookie* HTTP header line as follows in response to a URL request from a browser:

SET-Cookie: Cookie_Name=Cookie_Value; expire=Date; domain=Domain_Name; path=Path; Secure_Flag=boolean; Flag=boolean

- *Cookie_Name* and *Cookie_Value* have the actual information we want to keep in

the cookie. Optionally, we can store multiple *Cookie_Name* and *Cookie_Value* pairs in the same cookie, so they appear as one to the browser.

- *Date (Expire)* is the valid lifetime of the cookie. If the expiration date is set, the cookie will no longer be stored when the expiration date has been reached. By default, the cookie is set to expire when the browser is closed.

- *Domain_Name* is the host or domain where the cookie is valid. When a server is looking for the cookies for a particular Web site, a comparison of the *Domain_Name* is made with the actual Internet domain name of the host. For instance, this field could have "domain=.acme.com" as a domain name.

- *Flag* specifies whether or not all machines within a given domain can access the variable in the cookie. If true, all servers in the specified *Domain_Name* can use the cookie (and the browser will supply the cookie to all servers in that domain). If false, *Domain_Name* is interpreted as a host name, which is the only host to whom the browser will give that cookie.[1]

- *Path* sets the valid path at the domain for the cookie which restricts cookie usage within a site. Only pages in the path specified by this field can read the cookie. By default, the path is set to the path for the page that creates the cookie. The most general path is "/". The path "/foo" would match "/foodoc" and "/foo/index.html".

- If a cookie is specified *Secure*, the cookie will only be transmitted over secure communications channels, such as an SSL [WS96, DA99] connection.

All of the above fields are optional except *Cookie_Name* and *Cookie_Value*. Whenever a browser - which contains several cookies - sends an HTTP request for a URL to a Web server, the request includes only those cookies relevant to that server in the following form:

[1]Host names and IP addresses are susceptible to spoofing, so these are not very strong privacy controls.

Cookie: Cookie_Name1=Cookie_Value1; Cookie_Name2=Cookie_Value2; ...

According to the current HTTP state management mechanism, only the relevant *Cookie_Name* and *Cookie_Value* fields are sent to the Web server (selected by means of *Domain_Name* and *Flag* fields) by the browser. The remaining cookie fields, such as *Date*, *Domain_Name*, *Flag*, *Path*, and *Secure* fields, are not sent to the Web server. Instead they are used in the browser. For instance, a cookie will be deleted from the browser after its expiration date. Only those cookies relevant to that Web server (selected by means of *Domain_Name* and *Flag* fields) will be sent by the browser.

If the Web server finds any cookies that are applicable for the server, those cookies are used during this communication between the browser and the server. However, if the Web server does not find any cookies specified for it, either that server does not use cookies in the communication or the server creates new cookies for subsequent communication between the browser and the server. Furthermore, Web servers may update the contents of their cookies for any specific circumstance. The cookie-issuer is not important for cookie validation. In other words, a Web server can create cookies for other servers in the domain.

2.2 Secure Socket Layer (SSL)

SSL (Secure Socket Layer [WS96, DA99]) was introduced with the Netscape Navigator browser in 1994, and rapidly became the predominant security protocol on the Web. Since the protocol operates at the transport layer, any program that uses TCP (Transmission Control Protocol) is ready to use SSL connections. The SSL protocol provides a secure means for establishing an encrypted communication between Web servers and browsers.[2] SSL also supports the authentication service between Web

[2]In many cases, due to export restrictions from the United States, only weak keys (40 bits) are supported, but SSL technology is intrinsically capable of providing strong protection against network threats.

servers and browsers.

SSL uses X.509 certificates. Server certificates provide a way for users to authenticate the identity of a Web server. The Web browser uses the server's public key to negotiate a secure TCP connection with the Web server. Optionally, the Web server can authenticate users by verifying the contents of their client certificates.

Even though SSL provides secure communications between Web servers and browsers on the Web, it cannot protect against end-system threats. For instance, if a user receives attributes from the server over a secure channel, it does not mean that we can trust the user. In other words, once the user, let's say Alice, receives some attributes from the server over the secure channel, she is able to change the attributes or give them to other people, because SSL does not support the integrity service in the user's end system. Then, Alice (or the person impersonating Alice) can access the servers - which accept the attributes - using those forged attributes. However, as we will see later in this dissertation, SSL can be used as part of our solution to protect information on the Web.

2.3 Public-Key Certificate (X.509)

A public-key certificate [HFPS98, ITU93, ITU97] is digitally signed by a certificate authority (a person or entity) to confirm that the identity or other information in the certificate belongs to the holder (subject) of the corresponding private key. If a message-sender wishes to use public-key technology for encrypting a message for a recipient, the sender needs a copy of the recipient's public key. In contrast, when a party wishes to verify a digital signature generated by another party, the verifying party needs a copy of the public key of the signing party. Both the encrypting message-sender and the digital signature-verifier use the public keys of other parties. Confidentiality, which keeps the value of a public key secret, is not important to the service. However, integrity is critical to the service, as it assures public-key users that the public key used is the correct public key for the other party. For instance,

if an attacker is able to substitute his or her public key for the valid one, encrypted messages can be disclosed to the attacker and a digital signature can be forged by the attacker.

ITU (International Telecommunication Union) and ISO (International Organization for Standardization) published the X.509 standard in 1988, which has been adopted by IETF (International Engineering Task Force). X.509 is the most widely used data format for public-key certificates today and it is based on the use of designated certificate authorities (CAs) that verify that the entity is the holder of a certain public-key by signing public-key certificates. An X.509 certificate has been used to bind a public-key to a particular individual or entity, and it is digitally signed by the issuer of the certificate (certificate authority) that has confirmed the binding between the public-key and the holder (subject) of the certificate. An X.509 certificate consists of the following:

- version of certificate format
- certificate serial number
- subject's X.500 name (assigned by a naming authority)
- subject's public key and algorithm information
- validity period (beginning and end date)
- issuer's X.500 name (certificate authority)
- optional fields to provide unique identifiers for subject and issuer (Version 2)
- extensions (Version 3)
- digital signature of the certificate authority

The optional fields are available from Version 2 to make the subject name or the issuing certificate authority name unambiguous in the event the same name has

Certificate Content:

```
Certificate:
    Data:
        Version: v3 (0x2)
        Serial Number: 5 (0x5)
        Signature Algorithm: PKCS #1 MD5 With RSA Encryption
        Issuer: CN=data.list.gmu.edu, OU=LIST, O=GMU, C=US
        Validity:
            Not Before: Tue Feb 09 03:10:38 1999
            Not  After: Wed Feb 09 03:10:38 2000
        Subject: CN=admin.list.gmu.edu, OU=LIST, O=GMU, C=US
        Subject Public Key Info:
            Algorithm: PKCS #1 RSA Encryption
            Public Key:
                Modulus:
                    00:bc:d7:fc:4f:29:a4:29:a5:21:be:69:47:4d:55:db:37:50:
                    18:2b:6e:3e:b0:85:3e:0f:86:0f:be:58:2b:c9:d3:dc:bc:03:
                    bc:86:44:c4:f4:18:94:51:96:c6:f9:c5:db:b8:9d:88:5b:53:
                    b7:08:2f:86:64:cb:c2:7b:60:36:87
                Public Exponent: 65537 (0x10001)
        Extensions:
            Identifier: Certificate Type
                Critical: no
                Certified Usage:
                    SSL Client
            Identifier: Authority Key Identifier
                Critical: no
                Key Identifier:
                    a5:d7:08:bc:ff:07:bd:5a:d4:8d:d4:68:53:87:4b:af:81:90:
                    f0:4d
    Signature:
        Algorithm: PKCS #1 MD5 With RSA Encryption
        Signature:
            11:ca:b1:94:14:fb:67:a2:ad:90:f1:ee:88:24:a8:d3:fd:5c:75:34:fc:
            c1:68:23:e6:12:19:3a:5c:45:62:af:51:a0:2f:44:96:f8:2e:1f:75:9a:
            4b:9c:ed:2a:45:2e:db:c8:9c:56:1a:e1:75:0a:8e:bf:f8:44:b8:84:31:
            d8
```

Figure 2.2: An Example of X.509

been reassigned to different entities through time. Version 3 provides the extensions field for the incorporation of any number of additional fields into the certificate. These extensions make X.509v3 a truly open-ended standard with room to support diverse needs. It is possible for certificate issuers of interest to define their own extension types and use them to satisfy their own particular needs.

2.4 Attribute Certificate

The U.S. financial industry through the ANSI X9 committee developed attribute certificates [Far98b, Far98a], which have now been incorporated into both the ANSI X9.57 standard and X.509. An attribute certificate binds attribute information to the certificate's subject. Anyone can define and register attribute types and use them for his or her purposes. The certificate is digitally signed and issued by an attribute authority: furthermore, an attribute certificate is managed in the same way as an X.509 certificate. However, an attribute certificate does not contain a public key. Therefore, an attribute certificate needs to be used in conjunction with authentication services, such as another certificate (X.509) and SSL to verify the subject of the attribute. However, as we will see in this dissertation, our smart certificates have the ability to build attribute information into X.509v3 extensions rather, without losing effective maintenance, than putting this information into separate certificates.

2.5 Pretty Good Privacy (PGP)

PGP (Pretty Good Privacy [Zim95, Gar95]), a popular software package originally developed by Phil Zimmermann, is widely used by the Internet community to provide cryptographic routines for e-mail, file transfer, and file storage applications. A proposed Internet standard has been developed [CDFT98], specifying use of PGP. It uses existing cryptographic algorithms and protocols and runs on multiple platforms. It provides data encryption and digital signature functions for basic message protection services.

PGP is based on public-key cryptography, and defines its own public-key pair management system and public-key certificates. The PGP key management system is based on the relationship between key owners, rather than on a single infrastructure such as X.509. Basically, it uses RSA [RSA78] for the convenience of the public-key cryptosystem, message digests (MD5 [Riv92]) and IDEA [LM91] for the speed of process, and Diffie-Hellman for key exchange. The updated version supports more cryptographic algorithms.

Even though the original purpose of PGP is to protect casual e-mail between Internet users, we decided to use the PGP package for our implementation. The package is already widely used and satisfies our requirements, in conjunction with Web servers via CGI scripts [Her96] to protect cookies (described in Section 6.2. These cookies have the user' role information.

CHAPTER 3

OPERATIONAL MODELS

We identify two different approaches for obtaining a user's attributes on the Web, especially with respect to *user-pull* and *server-pull* models, in which each model has user-based and host-based modes [PS99b]. Each approach can be made to work, and we provide an analysis of their relative advantages and disadvantages. Basically, there are three components in both models: client, Web server, and attribute server. These components are already being used on the Web. Clients connect to Web servers via HTTP using browsers. The attribute server is maintained by an attribute authority and issues attributes for the users in the domain.

In this chapter, we focus on identifying the operational models for secure attribute services on the Web with tradeoffs between them. Detailed technologies (such as authentication, attribute transfer and protection, and verification) to support these models depend on the applications that are used.

3.1 User-Pull Model

In the user-pull model, the user pulls appropriate attributes from the attribute server and then presents them to the Web servers. Figure 3.1 shows a collaborational diagram in UML (Unified Modeling Language [BJR98, Qua98]) style notation for this model. We call this a user-pull model, because the user pulls appropriate attributes from the attribute server, in which attributes are issued for the users in the domain.

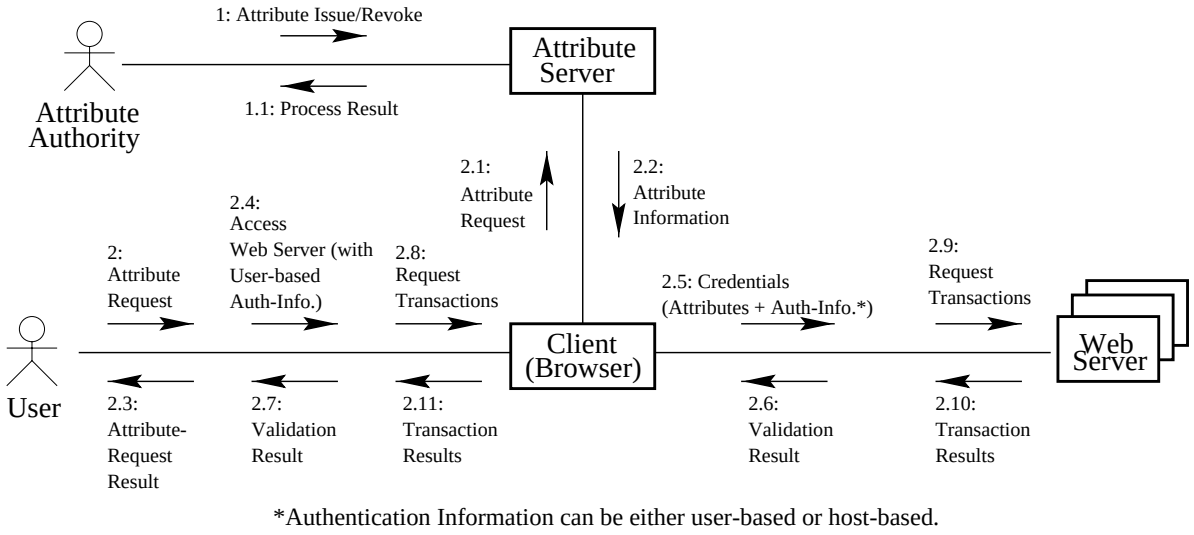


Figure 3.1: Collaborational Diagram for User-Pull Model

HTTP (HyperText Transfer Protocol) is used for user-server interaction with standard Web browsers and Web servers.

In user-pull host-based mode, a user, Alice, needs to download her attributes from the attribute server and store them in her machine (which has her host-based authentication information, such as a client certificate[1]). Later, when Alice wants to access the Web server, which requires proper authentication information and attributes, her machine presents that information to the Web server. After client authentication and attribute verification, the Web server uses the attributes for its purposes, such as access control, authorization, and electronic transactions. However, since this mode is host-based, it cannot support high user mobility, although it may support more convenient service than the user-based mode - which requires the user's cooperation (e.g., typing in passwords).

On the other hand, the user-pull user-based mode supports high user mobility.

[1] Optionally, we can use other host-based authentication information, such as IP numbers and Kerberos tickets.

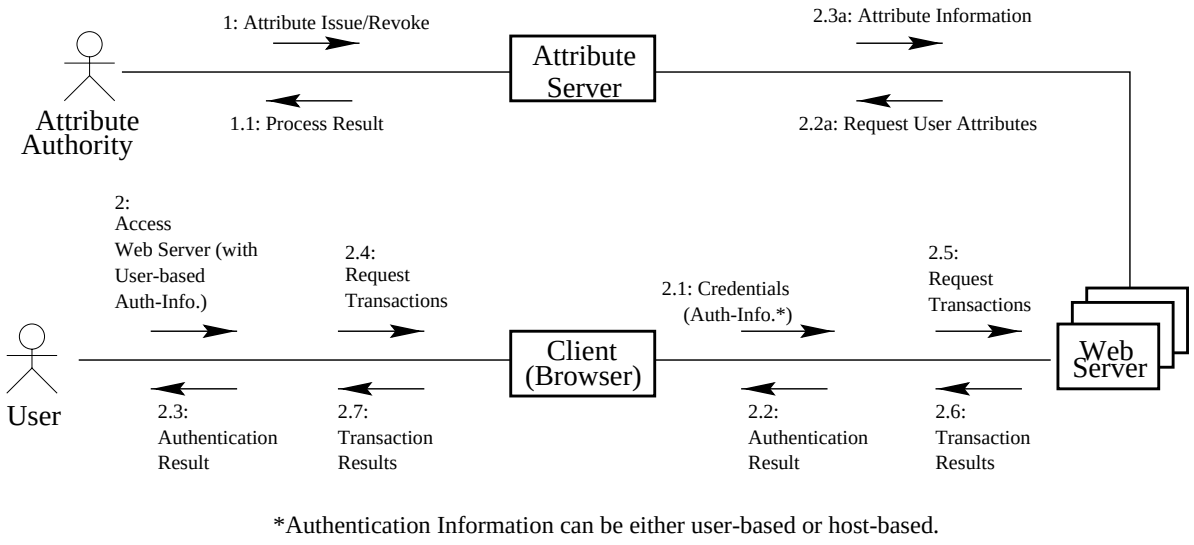


Figure 3.2: Collaborational Diagram for Server-Pull Model

A user, Alice, can download her attributes to her current machine from the attribute server. Then, Alice presents those attributes to the Web server along with her user-based authentication information, such as her passwords. After user authentication and attribute verification, the Web server uses the attributes for its purposes.

In this user-pull model, we must support the binding of attributes and identification for each user. For instance, if Alice presents Bob's attributes with her authentication information to the Web server, she must be rejected. We can use different mechanisms for binding attributes and user identifications.

3.2 Server-Pull Model

In the server-pull model, each Web server pulls appropriate attributes from the attribute server as needed and uses them for its purposes. Figure 3.2 shows a collaborational diagram in UML (Unified Modeling Language) style notation for this model. We call this a server-pull model, since the server pulls appropriate attributes from the attribute server. HTTP is used for user-server interaction with standard Web

browsers and Web servers. If the attribute server provides attributes securely, the Web server can trust those attributes and uses them for its purposes.

In this model, a user, Alice, does not need access to her attributes. Instead, she needs only her authentication information. In server-pull host-based mode, she presents host-based authentication information (e.g., her client certificate) to the Web server. The mechanism for obtaining attributes is transparent to the user, while limiting user portability. However, in server-pull user-based mode, Alice presents user-based authentication information, such as her passwords, to the Web server. This supports high user portability, while it requires the user's cooperation (e.g., typing in passwords). After client(user) authentication, the Web server downloads the corresponding attributes from the attribute server and uses them for its purposes, such as access control, authorization, and electronic transactions.

CHAPTER 4

COOKING SECURE COOKIES ON THE WEB

Since there is no concept of a "session" in HTTP, Web servers and browsers use cookies [KM97, KM98a, KM98b, Lau98] to capture information for subsequent communications, thus providing continuity and state across HTTP connections. Technically, it is not difficult to equip a cookie to carry any relevant information. For instance, a merchant Web server could use a cookie that contains the user's name and credit card number. This is convenient for users, since they do not have to read lengthy numbers from their cards and key these in for every transaction. However, it is not safe to store and transmit sensitive information in cookies, as cookies are insecure.

Cookies are stored and transmitted in clear text, which is readable and can be forged. In this chapter, we identify and discuss several techniques to render it secure to carry and store sensitive data in cookies. We show how secure cookies support secure attribute services between existing Web servers and browsers, and identify examples of electronic commerce and business applications for use of this technology. Secure cookies are constructed by using familiar cryptographic techniques, such as message digests, digital signatures, message authentication codes, and encryption. The novelty lies in the manner in which these techniques are applied to implement secure cookies and in the Web services to which secure cookies are applied. A notable aspect is that secure cookies can be issued by one Web server for use by another. This facilitates secure attribute services on the Web by means of secure cookies. The secure cookie mechanism is a inherently user-pull model (described in Chapter 3),

since users need to obtain their attributes from an attribue server and present them to Web servers by means of secure cookies on the Web.

Our techniques have varying degrees of security and convenience for users and system administrators. It is our well-founded belief that there is no such thing as absolute security. Hence, these techniques are not intended to be foolproof but rather to provide a reasonable degree of security. We will consider vulnerabilities of our techniques as we go along. An equally well-founded belief is that a one-size-fits-all approach is rarely effective in systems design. Hence, the need exists for a variety of techniques for secure cookies providing different tradeoffs between security and convenience for end users, system administrators, and application developers.

Secure cookies provide three types of security services: authentication services, integrity services, and confidentiality services. Authentication services verify the owner of the cookies. Integrity services protect against the threat that the contents of the cookies might be changed by unauthorized modification. Finally, confidentiality services protect against the values of the cookies being revealed to an unauthorized entity. In this chapter, we describe how to transform regular cookies (which have zero security; see Figure 2.1), into secure cookies that give us the basis for providing security services.

4.1 Security Concerns in Cookies

Cookies are often used by Web servers to tag visitors, and to identify who they are and what the status is for each visitor. For instance, if a merchant Web site has a customer database that contains customers' information, such as names, addresses, payment histories, merchandising histories, and credit card numbers, the Web site creates and uses cookies to store pointers to individual customer records in the customer database. Since cookies are readable and can be easily forged, it is reasonable to store a simple ID number of a customer in a cookie rather than all of the customer's information. This ID number is exposed in a cookie without exposing the actual customer data.

However, even this ID can be forged or used by someone else.

Cookies are electronic notes for Web servers stored on the browser side. Some people may be concerned that any random site could read the cookies on a user's hard disk, or that cookies could be used to steal information from a user's hard disk. Fortunately, there is no way a cookie can examine the contents of the user's hard drive, nor can it release the user's private data on the Web, because a cookie is written in a text file stored in a user's machine. Therefore, if cookies do not carry any sensitive information about the user, or the sensitive information is not in plain text (for instance, if cookies carry encrypted text of sensitive information), we can achieve confidentiality of data transmitted in cookies.

We mention that one of the main concerns about cookies in the popular press has been privacy, because cookies allow Web servers to track a user's browsing behavior. These privacy concerns are outside the scope of this work. Our focus is on the integrity, authenticity, confidentiality and authorization for non-anonymous Web transactions.

4.2 Security Threats to Cookies

However, what if a malicious user were to change the contents of the cookies? What if a malicious user were to use the cookies of other people? What if a malicious user were to collect cookies from other people and extract some sensitive information, such as credit card numbers or passwords? Actually, these attacks are easy to carry out, since cookies are stored in plain text somewhere on the user's hard disk. A user, say Alice, can change the contents of her cookies, such as ID numbers to a Web site, and access the Web site with the forged cookies. Alice can copy cookies from Bob's machine to her machine, and easily can impersonate Bob in the Web sites that accept Bob's cookies. Furthermore, if Alice can impersonate a site that accepts Bob's cookies, or if she penetrates one such site, Bob's browser sends Alice all the cookies from that domain. Then, Alice can use the harvested cookies for

all other sites within the domain. Therefore, cookies are not the appropriate place to store sensitive information, including passwords, credit card numbers, purchase authorization numbers, and so on.

We distinguish three types of threats to cookies: network threats, end-system threats, and cookie-harvesting threats. Cookies transmitted in clear text on the network are susceptible to snooping (for subsequent replay) and to modification by *network threats*. Network threats can be foiled by use of the Secure Sockets Layer (SSL) protocol, which is widely deployed in servers and browsers. However, SSL can only protect cookies while they are on the network. Once the cookies are sent to the browser's end system, they reside on the hard disk or memory in clear text. Such cookies can be trivially altered by users and can be easily copied from one computer to another, with or without connivance of the user on whose machine the cookie was originally stored. We call this the *end-system threat*. The ability to alter cookies allows users to forge authorization information in cookies and to impersonate other users. The ability to copy cookies makes such forgery and impersonation all the easier. Additionally, if an attacker collects cookies by impersonating a site that accepts cookies from the users (who believe that they are communicating with a legitimate Web server), later he can use those harvested cookies for all other sites that accept those cookies. We call this the *cookie-harvesting threat*. These attacks are all relatively easy to carry out and certainly do not require great hacker expertise. Now, we will describe how to protect cookies from these threats in the following section.

4.3 User Authentication for Cookies

It is already possible to use cookies on the Web between existing Web servers and browsers. However, a malicious user can simply snatch the cookies of other people and impersonate the real owner of the cookies in the server that is accepting those cookies. To solve this problem, we identify four possible authentication methods for cookies: address-based authentication, password-based authentication, Kerberos-

	Domain	Flag	Path	Cookie_Name	Cookie_Value	Secure	Expire
IP_Cookie	acme.com	TRUE	/	IP_Cookie	129.174.100.88	FALSE	12/31/99
Pswd_Cookie	acme.com	TRUE	/	Pswd_Cookie	hashed_password	FALSE	12/31/99
KT_Cookie	acme.com	TRUE	/	Kerberos_Ticket	{Alice, $K_{c,s}$}K_s	FALSE	12/31/99
Sign_Cookie	acme.com	TRUE	/	Sign_Cookie	Signature_of_Alice	FALSE	12/31/99

Figure 4.1: Authentication Cookies

based authentication, and digital-signature-based authentication. Figure 4.1 shows the authentication cookies. Note that we need to use one or some of those authentication cookies from Figure 4.1 with regular cookies in Figure 2.1. The choice of an authentication cookie depends on the given situation.

4.3.1 Address-Based Authentication

We use the IP_Cookie, which grabs the IP address of the user's machine for address-based authentication to protect malicious users from impersonating the original owner of the cookies. Since the IP address is one of the environment variables for a user on the Web[1], it is not difficult for a server to obtain the user's IP address and put it into the IP_Cookie by internal procedures, such as CGI scripts in the server. Now, whenever Alice tries to access the server that accepts the cookies, the server checks first to see if Alice's current IP address is the same as the one in the IP_Cookie Alice sent to the server. If they are identical, the server believes that Alice is the real owner of those cookies.

[1]Every process or program on the Web has an environment of data with which it begins. The environment is described by environment variables such as: REMOTE_ADDR, REMOTE_HOST, REQUEST_METHOD, SERVER_NAME, SERVER_PORT, SERVER_PROTOCOL, QUERY_STRING.

It is very convenient for users to use their IP addresses as simple authentication information, since users do not need to enter their authentication information during communication between servers and browsers. However, in some cases, using the user's IP address for authentication is not a good solution. For example, what if Alice's IP address is dynamically assigned to her machine whenever she connects to the Internet? In this case, even though Alice is using the same machine as before, the cookies that Alice received in a previous internet connection are not valid any more once the IP address is changed. Furthermore, if Alice's domain uses a proxy server to reach Bob, an attacker can collect Alice's cookies by the cookie-harvesting and can easily impersonate Alice to Bob through the same proxy server that provides the same IP numbers to users. In addition, we cannot avoid the IP spoofing problem whenever we use IP addresses for some purposes on the Internet.[2] Therefore, to avoid the above problems in such cases, we need to use one of the other authentication cookies described below.

4.3.2 Password-Based Authentication

To support users who use dynamic IP addresses or proxy servers, and to avoid the IP spoofing, we can use password-based authentication. In other words, if the server grabs Alice's password first and puts the hash of the passwords into a cookie, call it Pswd_Cookie, servers can authenticate the owner of the cookie. Alice is required to type her password for verification whenever she tries to access other servers that accept those cookies.[3] If the hash of the password she entered is the same as the one in her Pswd_Cookie, then the server believes Alice is the real owner of those cookies. Alternatively, servers can use encrypted passwords instead of the hash of the password in the Pswd_Cookie to authenticate the owner of the cookie (the detailed encryption process is described in the following subsection). As a result, no one but

[2]Strong solutions to IP spoofing require use of IPSEC [KA98].

[3]The passwords will be transmitted from browser to server and should be protected on the network by means of SSL.

Alice can use those cookies on the Web. However, this mechanism requires users to enter their passwords for authentication whenever they connect to the site, but using an IP_Cookie is transparent to users. Furthermore, it is vulnerable to dictionary attacks, since hashed or encrypted passwords are revealed in the Pswd_Cookie.

4.3.3 Kerberos-Based Authentication

We can also use Kerberos[4] [SNS88, Neu94] with cookies for mutual authentication. We assume the reader is familiar in general terms with the Kerberos protocol. An HTTP adaptation of the Kerberos protocol is shown in Figure 4.2. Kerberos is a secret-key based authentication service in a network. To use Kerberos with cookies for authentication, the user needs additional browser software to replace the value of the cookie (TGT_Cookie), which contains a ticket-granting ticket (TGT), and generates authenticators ($\{\text{timestamp}\}S_A$, $\{\text{timestamp}\}K_{C,S}$) in the TSK_Cookie and the TSS_Cookie respectively. Furthermore, we need to modify the Kerberos protocol slightly to enable the Kerberos ticket cookie, KT_Cookie, to work with other secure cookies, which particularly support the integrity of cookies.

Figure 4.2 shows how Alice logs into Bob through the KDC (Key Distribution Center) with a KT_Cookie. The KDC shares a secret key with each principal in the domain. During the initial login, Alice asks the KDC for a session key, S_A, and a ticket-granting ticket (TGT), which is encrypted by the KDC's master key, K_{KDC}. Then, Alice receives $\{\text{TGT}, S_A\}K_C$ (encrypted with Alice's secret key, K_C), stored in the TGT_Cookie from the KDC.

Alice decrypts the encrypted S_A and the TGT using her secret key, K_C. Then, Alice replaces the value of the TGT_Cookie with the TGT and removes $\{\text{TGT}, S_A\}K_C$ from her machine. Practically, it is possible for Alice to create another cookie containing the TGT. However, keeping $\{\text{TGT}, S_A\}K_C$ in Alice's machine for a long time

[4]It also is possible for authentication to be based on use of RADIUS [RRSW97] and similar protocols. Our focus in this research is on techniques that make secure cookies self-sufficient rather than partly relying on other security protocols, which is always possible.

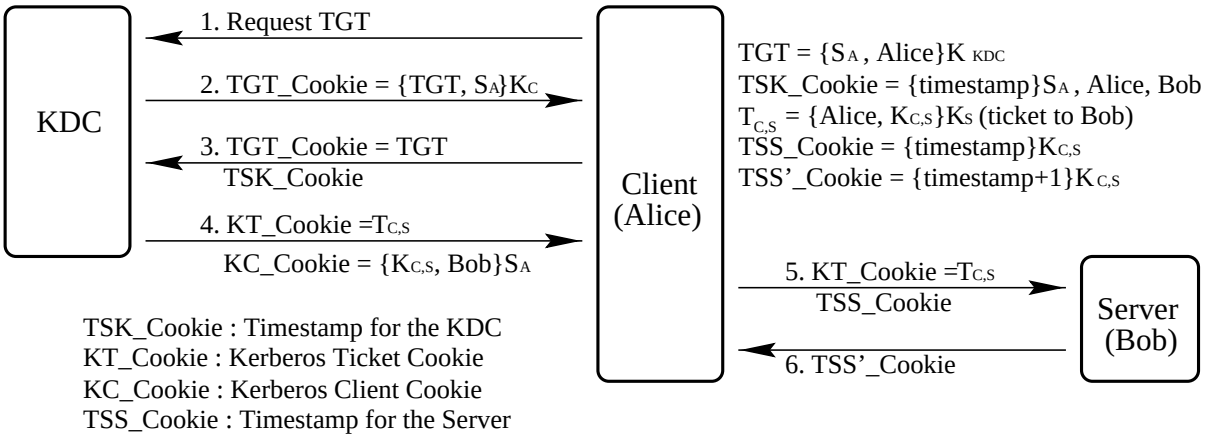


Figure 4.2: Kerberos-based Authentication with Cookies

is not a good idea from a security point of view, once Alice does not need it any more.

When Alice later needs to access a remote server (say Bob), she connects to the KDC with the TGT_Cookie, along with the TSK_Cookie, which is generated by Alice's machine to carry an authenticator ($\{\text{timestamp}\}S_A$) and the name of the remote server, Bob. The KDC verifies the timestamp and decrypts the TGT to discover S_A. At that point, the KDC creates both the Kerberos client cookie, KC_Cookie, containing $\{K_{C,S}, \text{Bob}\}S_A$ for Alice, and the Kerberos ticket cookie, KT_Cookie, containing $T_{C,S}$ (encrypted with Bob's secret key) for Bob. As a result, the KDC sends Alice $\{K_{C,S}, \text{Bob}\}S_A$ and $T_{C,S}$ separately.

According to the original Kerberos protocol, the KDC is supposed to encrypt those values all together and send $\{K_{C,S}, T_{C,S}, \text{Bob}\}S_A$ to Alice in one cookie. However, this approach gives us a limitation for using the KT_Cookie with other secure cookies. For instance, if we need to use a KT_Cookies with regular cookies in Figure 2.1 for authentication, we may also consider the integrity of the cookies that can be supported by another secure cookie, Seal_Cookie (described in section 4.4). By the original Kerberos protocol, users must change the contents of the cookie or generate another one to make a cookie that has the ticket to Bob ($T_{C,S}$). This obviously

conflicts with the integrity of the cookies. Therefore, the KDC is required to create two separate cookies (KC_Cookie and KT_Cookie) and send them to Alice.

When Alice connects to Bob, the KT_Cookie and the TSS_Cookie - which is generated by Alice's machine containing an authenticator ($\{\text{timestamp}\}K_{C,S}$) - are sent by Alice to Bob. To provide mutual authentication, Bob responds to the authenticator. Now, Alice believes that she is connecting to Bob, since Bob was able to decrypt the ticket in the KT_Cookie, which meant he knew $K_{C,S}$ encrypted with K_S.

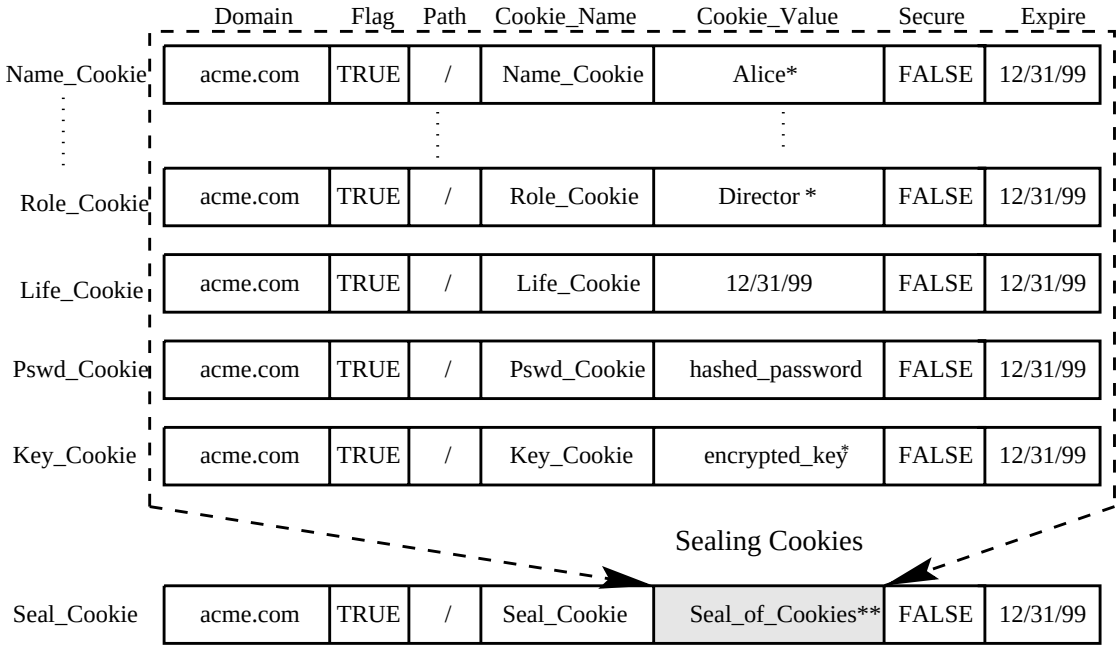
4.3.4 Digital-Signature-Based Authentication

If servers know users' public keys, the digital signature technologies, such as DSA [Fed91, Fed94] and RSA [RSA78] can be used to authenticate users with cookies. To use this method, the user needs additional browser software to generate a cookie that contains a signed timestamp. For instance, when Alice needs to access a remote server (say, Bob), who knows Alice's public key, Alice's machine generates a timestamp and creates the Sign_Cookie shown in Fig 4.1, which has Alice's digital signature (signed with her private key) on the timestamp. When Alice connects to Bob, he receives the Sign_Cookie from Alice and verifies the signature with her public key.[5]

4.4 Providing Integrity to Cookies

There are also integrity problems with cookies. For instance, if an attacker wants to impersonate Alice, he can copy the IP_Cookie from Alice and edit it with his IP number, and later impersonate Alice in a Web server that accepts the IP_Cookie. Figure 4.3 shows a set of secure cookies and Figure 4.4 shows how the secure cookies are used on the Web. The regular cookies (Name_Cookie and Role_Cookie) and the authentication cookie (Pswd_Cookie) have the user's attributes and authentication

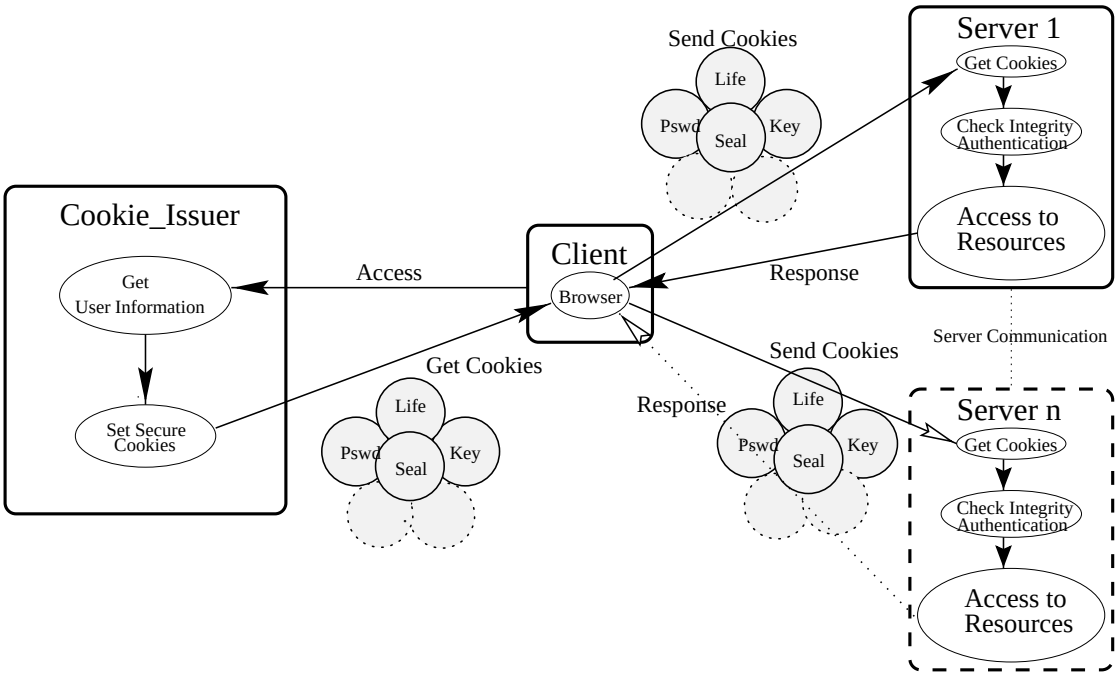
[5]This technique requires each user to have a public-key certificate. With this assumption, one could conceivably use this certificate to do user-to-server authentication at the SSL layer instead.



* Sensitive fields can be encrypted in the cookies.
** Seal_of_Cookies can be either MAC or a signed message digest of cookies.
Note: Pswd_Cookie can be replaced with one of the other authentication cookies in Figure 4.1

Figure 4.3: A Set of Secure Cookies on the Web

information. The Life_Cookie is used to hold the lifetime of the secure-cookie set in its *Cookie_Value* field and enables the Web server to check the integrity of the lifetime of the secure-cookie set. Even though only the relevant *Cookie_Name* and *Cookie_Value* fields will be sent to the Web server (selected by means of *Domain_Name* and *Flag* fields) by the browser, the Web server can check the integrity of the other fields in the cookies under a certain policy. For example, if the policy (established between the cookie-issuing server and the Web servers in the domain) presets the values of *Domain_Name*, *Flag*, *Path*, and *Secure* fields with "acme.com," "TRUE," "/," and "FALSE" respectively, the Web server uses those values to check the integrity of the cookies. However, it is not a good idea to preset an expiration date for all the cookies under a policy in the domain, since each secure-cookie set may require a different



Pswd_Cookie can be replaced with one of the other authentication cookies in Figure 4.1.

Figure 4.4: How to Use Secure Cookies on the Web

lifetime. To solve this problem, the Life_Cookie is used.[6] If it is necessary, we can include other regular cookies in the secure-cookie set, holding specific information to be protected according to applications. The Key_Cookie is optionally used to facilitate encryption of sensitive data as will be described in Section 4.5. In some cases, if we do not need the confidentiality of information in cookies - for example, when we use cookies without sensitive information - we do not need an encryption mechanism. Namely, we do not need the Key_Cookie. Finally, the Seal_Cookie is used to determine if the contents of the cookies have been altered. The contents of the Seal_Cookie depend on the cryptographic technologies used for the cookie. Basically, there are two solutions: public-key-based solution and secret-key-based solution. The

[6] A cookie is deleted from a user's machine automatically after its expiration date. This affects the integrity of the secure-cookie set. Therefore, it is efficient to set the same expiration date for all the cookies that are also stored in the *Cookie_Value* field of the Life_Cookie.

key distribution between servers can be achieved by key agreement algorithms, such as RSA [RSA78] or Diffie-Hellman [DH76, DH97].

4.4.1 Public-Key-Based Solution

In the case of using public-key technology, the cookie-issuing server creates a message digest from the rest of the cookies using a message digest algorithm, such as MD5 [Riv92] or SHA [Fed95], then signs the message digest using its private key and puts the signature in the Seal_Cookie.

When the user connects to a Web server, it gets a relevant set of secure cookies from the browser. The Web server then verifies the signature in the Seal_Cookie using the public key of the cookie-issuing server and the values set by the policy (between the cookie-issuing server and Web servers). If the integrity verification is successful, then there has been no alteration in the cookies. If the cookies are valid, the Web server trusts the cookie values and uses them for its purposes.

In the public-key-based solution, only the server, which has the private key for the Seal_Cookie, can update the contents of the secure cookies for any specific circumstance. For instance, when we need to update the contents of the secure cookies, the user has to go back to the original cookie-issuing server - which has the private key - to update the cookies, because the updated cookies need to be signed with the cookie-issuing server's private key.[7]

4.4.2 Secret-Key-Based Solution

In the case of using the secret-key cryptography, the cookie-issuing server creates a message authentication code (MAC) from the rest of the cookies by using a key-dependent one-way hash function, such as HMAC [BCK96], and puts it in the Seal_Cookie. When the user connects to a Web server - which accepts the cookies - the server ob-

[7] If the private key is shared among these servers, it is possible to update the secure cookies at multiple sites.

tains all the relevant cookies for itself from the browser. Assuming that the Web server has a secret key shared with the cookie-issuing server, the server creates a message authentication code (MAC) from the cookies and the values set by the policy, and compares it with the one in the user's Seal_Cookie. If the two MACs are identical, the Web server believes that there has been no alteration in the cookies.

If we use the secret-key-based solution for secure cookies, any server that has the shared secret key can update the contents of the cookies for any specific circumstance. For instance, a Web server - which has the shared secret key with the cookie-issuing server - can extend the expiration date of the cookies. Therefore, the user does not have to go back to the original cookie-issuing server (which created the cookies) to update the cookies.

4.5 Providing Confidentiality to Cookies

Additionally, let us consider how to prohibit other people, or perhaps even the cookie owner, from reading sensitive information in the cookies. To do so, sensitive information, such as names, roles, and credit card numbers, can be encrypted by the Web server and stored in cookies. We use the Key_Cookie, shown in Figure 4.3, to store an encrypted session key, which was used to encrypt sensitive information of other cookies. The session key can be encrypted either by the server's public key or by the server's secret key. We may encrypt the contents of the cookies with the server's secret key or public key directly (we do not need the Key_Cookie in this case). However, for more secure services, we recommend using a session key for confidentiality services with cookies. In the case of using the public-key cryptography for encryption, we recommend using separate public-key pairs for encryption and digital signature, since servers may share the private key for information decryption while keeping the private key for digital signature secret.

When a Web server receives secure cookies, including the Key_Cookie, the server decrypts the *Cookie_Value* of the Key_Cookie by using its master key - private

key or secret key - to get the session key. The secret key is shared with other Web servers and the session key was used to encrypt sensitive information in other cookies. At this point, the server decrypts and reads the encrypted information in the cookies using the session key. If the contents of the cookies were encrypted by the server's secret key or public key directly, the server decrypts the encrypted information in the cookies using the corresponding key.

4.6 Secure Cookies and SSL

A cookie can be transmitted over SSL [WS96] if the *secure* flag is on in the cookie. Even though SSL provides secure communications between Web servers and browsers on the Web, it cannot protect the end-system threats. For instance, if a user receives cookies from the server over a secure channel, it does not mean that the cookies are securely stored in the user's machine. In other words, once Alice receives some cookies from the server over the secure channel, she is able to change the contents of the cookies or give them to other people. Then, Alice or the person, who is impersonating Alice, can access the server (which accepts the cookies) using those forged cookies. However, SSL can be used as part of our solution to protect information on the Web. For example, users can securely transmit their passwords to Web servers over SSL.

It is also our contention that SSL can be used as part of our solution to make systems more secure and efficient on the Web. Suppose we have hundreds of Web servers in an organization, requiring integrity, authentication, and confidentiality services in cookies. It is not a good idea to allow individual Web servers to share a secret key with others in the domain, since this increases the likelihood of exposing the secret key. To support more secure service, several verification servers can be used that have the secret key shared with other verification servers to verify, decrypt, or update cookies issued by cookie-issuing servers in the domain. As a result, the remaining servers do not have the shared secret key. When a Web server receives a set of secure cookies from a user, the server asks one of those verification servers if the cookies are valid, forwarding those cookies. Then, the verification server verifies

Effect	SSL	Auth			Seal	Auth + Seal	Auth + Key+Seal
		IP	Pswd*	Sign			
Network Threats	O	X	X	X	X	O	O
End System Threats	X	X	X	X	X	O	O
Cookie-Harvesting Threats	O	X	O	O	X	DA	DA
Information Hiding	No	No	No	No	No	No	Yes
Additional Client Software	NR	NR	NR	R	NR	DA	DA
IP Spoofing	O	X	NO	NO		DA	DA

O = can solve the problem; X = cannot solve the problem; R = required;
NR = not required; DA = depends on the authentication cookie
* Passwords are protected by SSL on the network.

Table 4.1: Summary of the Recipes for Secure Cookies

the cookies and sends the result to the Web server in plain text over the secure channel achieved by SSL. Furthermore, if it is necessary, the verification server can decrypt the encrypted values in the cookies, or update information in the cookies. Finally, the Web server trusts the decrypted and verified information received from the verification server over an SSL and uses them for its purposes.

4.7 Comparison of the Recipes and Effects of Secure Cookies

A summary of the recipes for secure cookies on the Web is shown in Table 4.1. Regular cookies are susceptible to network threats, end-system threats, and cookie-harvesting threats. Actually, there is no security in regular cookies. Since SSL provides server-to-client authentication (optionally mutual authentication is possible by requiring client-to-server authentication) and data encryption on the Web, SSL can protect cookies from network threats, IP spoofing, and cookie-harvesting threats. However, it cannot protect cookies from the end-system threats. One of the authentication cookies - IP_Cookie, Pswd_Cookie, KT_Cookie, or Sign_Cookie and Seal_Cookie can be used to provide authentication and integrity services to cookies, respectively. If authentication is dependent on IP addresses, IP spoofing remains a threat. Further-

more, the IP_Cookie cannot protect cookies from cookie-harvesting threats if users use a proxy server. On the other hand, if servers use the Pswd_Cookie, carrying the hash of the user's passwords; the KT_Cookie, carrying the Kerberos ticket; or the Sign_Cookie, carrying the user's digital signature, authentication has neither IP spoofing exposure nor cookie-harvesting threats. However, using Pswd_Cookie requires users to enter their authentication information for each server in the domain and requires SSL to protect the passwords on the network and to achieve server-client authentication to prevent server spoofing. Use of the IP_Cookie is transparent to users, and using the KT_Cookie supports mutual authentication between clients and servers. However, using either the KT_Cookie or the Sign_Cookie requires additional browser software to deal with ticket-granting tickets and authenticators. Additionally, if cookies have some sensitive information that is need to be encrypted by a session key, the Key_Cookie can be used with other secure cookies to store the encrypted session key.

4.8 Applications of Secure Cookies

In this section, we introduce several applications of secure cookies. The followings are representative samples, and many other applications can be imagined.

4.8.1 User Authentication

Web servers can simply use secure cookies to authenticate their visitors. One or multiple authentication cookies (IP_Cookie, Pswd_Cookie, KT_Cookie, or Sign_Cookie), depicted in Figure 4.1, can be used to support user authentication service in the Web server in conjunction with Seal_Cookie, which supports the integrity service for the cookies. Detailed procedures for creating and using those cookies and comparisons are described in Section 4.3.

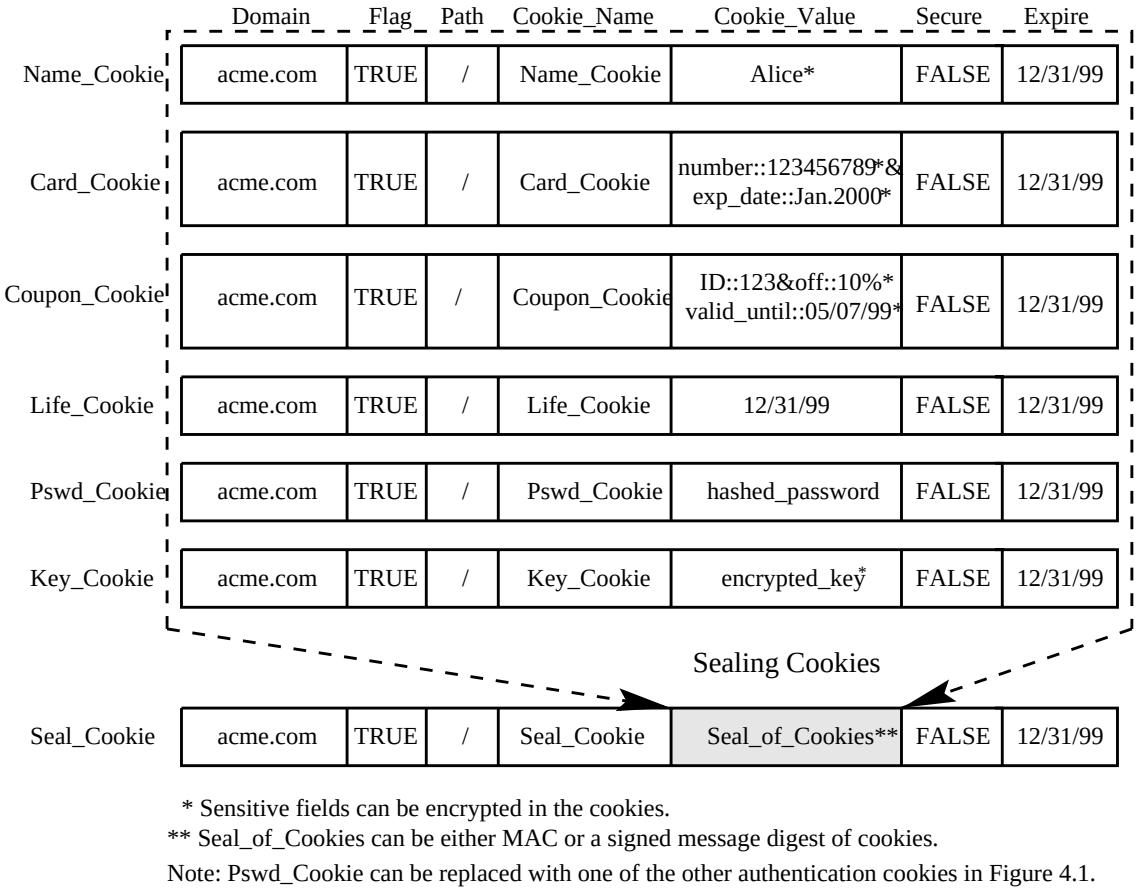


Figure 4.5: An Example of Secure Cookies for Electronic Transactions

4.8.2 Electronic Transactions

In Figure 4.5, the Card_Cookie and Coupon_Cookie have multiple pairs of *Cookie_Name* and *Cookie_Value* in their *Cookie_Value* fields. Intuitively, a merchant site should set the Card_Cookie to expire before the actual expiration date of the credit card. For more convenient services, the merchant can issue special tokens for customers, such as electronic coupons, which contain the coupon's ID number and discount information. For instance, if Alice received an electronic coupon along with other secure cookies, she can use it by the coupon's valid date in the merchant site. In this case, the merchant site needs to keep a record for the coupon by the ID to prevent replay usages

of the same coupon.

4.8.3 Eliminating Single-Point Failure

Because of the regular cookies' insecurity, a merchant site usually sets a cookie to hold a pointer, such as an ID number, which is a key field of the customer-information database in the site. However, this implies that a customer-information database must be maintained in a server. One of the disadvantages of this method is that if the server holding customers' information is penetrated by an attacker, all the customers' information, such as credit card numbers, preferences, addresses and other sensitive information in the server, is open to the attacker. Furthermore, if a domain has multiple servers with multiple customer-information databases, maintenance and synchronization of this information is burdensome. There are also significant privacy concerns about data stored by servers, since such data can easily be misused. Users may feel more comfortable with servers that pledge not to maintain such data.

Secure cookies can solve these problems, especially in the electronic commerce field. If a merchant site creates and uses the set of secure cookies shown in Figure 4.5, the site does not need to have a customer-information database unless the site needs to track customers' access histories. This is because each customer's sensitive information is distributed and stored securely in the customer's hard disk via secure cookies. The secure cookies provide a more secure environment by eliminating customer-information databases that can cause a single point failure. Furthermore, the merchant can reduce the cost for the maintenance of customer-information databases.

4.8.4 Pay-Per-Access

Today, many pay-per-access sites provide various information services, such as performances, games, movies, and other special events, in the Web environment. Therefore, it is obvious that we need secure mechanisms to sell, buy, and use tickets to such sites on the Web.

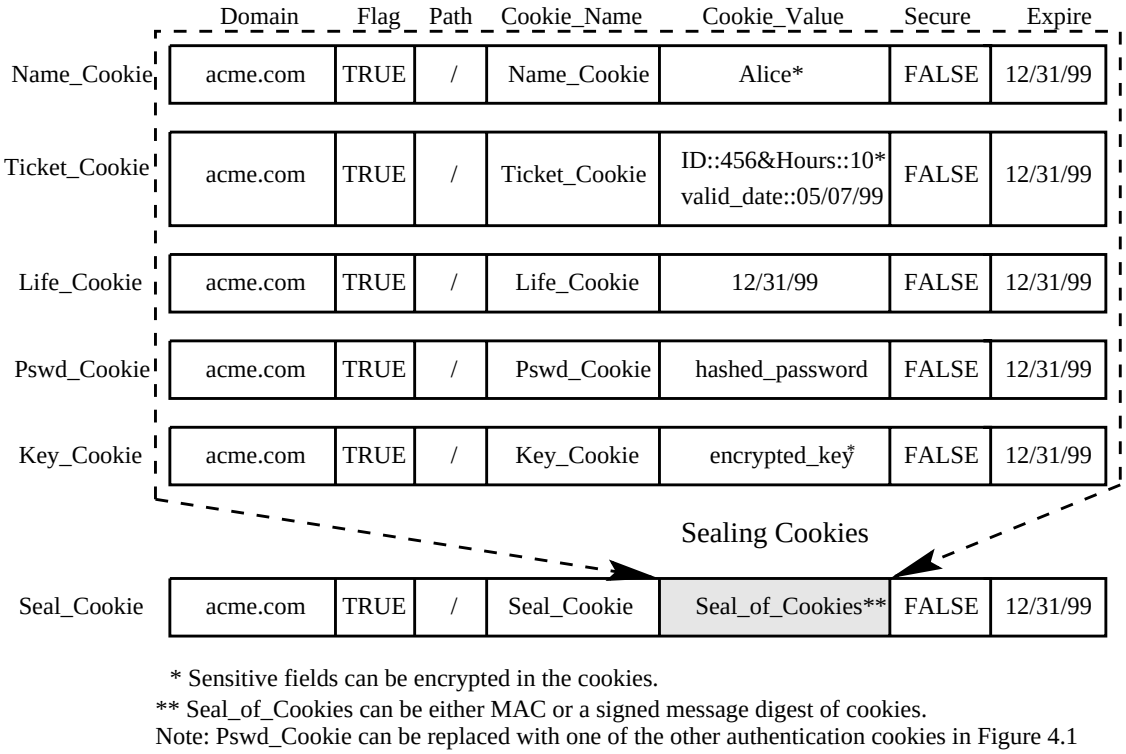


Figure 4.6: An Example of Secure Cookies for Pay-Per-Access

Secure cookies can be used to solve this problem. Suppose Alice wants to buy access to a pay-per-access site. After Alice pays for access, the server has to give her a token that allows her to access the site until the ticket expires. If Alice receives a cookie, let's call it Ticket_Cookie (which contains her access limit and valid date) along with other secure cookies, she can access the site as long as the Ticket_Cookie is valid. No one but Alice can use her secure cookies. Figure 4.6 shows an example of a set of secure cookies for Pay-Per-Access sites on the Web.

The access limit in the Ticket_Cookie is controlled either by time or by the number of connections. For instance, if a merchant site issues a 10-hour (or 10-use) ticket to Alice, the site allows Alice to use its resources up to 10 hours (or 10 uses) until the valid date of the Ticket_Cookie.

A merchant site could control a customer's access limit by simply updating

the content of the Ticket_Cookie with the customer's residual access limit (hours or number of uses). However, this is not a good idea, since a smart and malicious customer can access the site beyond his or her access limit. For instance, if Alice uses a copy of a whole set of her initial secure cookies, including the Ticket_Cookie, then she can have unlimited access to the site until the cookie set is valid because the merchant site has no idea about customers' replays.

To prevent this kind of replay attacks, a merchant site needs to keep information about tickets - such as accumulated usage of each ticket - at least until the Ticket_Cookie becomes invalid. This does not mean that the merchant site needs to keep all the customer information. In other words, only the ticket ID and its accumulated usage are required to be tracked by the merchant site. In this case, merchant sites do not need to update the contents of the cookies by replay attacks. For instance, if Alice's accumulated access usage becomes more than the access limit denoted in her Ticket_Cookie, then the merchant site rejects Alice's request.

4.8.5 Attribute-Based Access Control on the Web

If a user's attribute information is stored in cookies securely as shown in Figure 4.3, Web servers can use those secure cookies for attribute-based access control on the Web. Since all the attribute information is protected from possible security threats on the Web as well as in end-systems, Web servers can verify and trust the attribute information in the secure cookies and use it for their purposes.

4.9 Summary

In this Chaper, we have described how to make and use secure cookies. The secure cookie mechanisms we have described use familiar cryptographic building blocks and leverage existing technology. We have also identified a variety of diverse applications for secure cookies. Using secure cookies on the Web can solve the security problems of regular cookies as well as the stateless problem of HTTP on the Web. Secure cookies

can be used between existing Web servers and browsers. Furthermore, using secure cookies is transparent to users and can eliminate customer-information databases that cause a single point failure. A proof-of-concept implementation is described in Chapter 6.

CHAPTER 5

EXTENDING X.509 FOR SECURE ATTRIBUTE SERVICES

Public-key infrastructure (PKI) has been recognized as a crucial enabling technology for security in large-scale networks. To support PKI, X.509 [HFPS98, ITU93, ITU97] certificates have been widely used. The basic purpose of X.509 certificates is simply the binding of users to keys. An X.509 certificate contains extension fields, which can be defined by anyone. Therefore, it is possible for certificate authorities to define and use their own extension types to satisfy their own particular needs. Each extension type needs to be registered by having an object identifier assigned to it. Even though X.509 has the ability to be extended, the application of the extensions of X.509 for secure attributes has not yet been precisely defined.

In this chapter, we describe how to extend and use existing X.509 certificates for secure attribute services on the Web [PS99b]. Our discussion of the extended X.509 certificates, which we have named *smart certificates*, also focuses on maintaining their compatibility with standard protocols, such as SSL, and effective maintenance. Furthermore, we discuss possible applications of smart certificates for electronic commerce and business.

5.1 Designing Smart Certificates by Extending X.509

We have extended X.509 certificates with several sophisticated features without losing effective maintenance; they support short-lived lifetime and multiple CAs, contain attributes, provide postdated and renewable certificates, and provide confidentiality. Selection of these new features depends on the applications that are used. Smart certificates can support both user-pull and server-pull models (described in Chapter 3).

5.1.1 Support for Short-Lived Certificates

Typical validity periods for X.509 certificates are several months or even years. To support user mobility, users should be able to download both the certificate and the private key to the software in different environments. This service may leave copies of private keys behind; therefore, the longer-lived certificates have a higher probability of being attacked. If, however, the validity periods for certificates are measured in hours, the user portability can be provided securely, since the copies of the corresponding private key expire shortly. Additionally, we do not need a revocation scheme (CRL) for the certificates, which is responsible for the complexity and cost of the public key infrastructure. The detailed description of short-lived certificates is available in [HS98].

5.1.2 Containing Attributes

It is strongly recommended that public-key pairs used for any purpose be updated periodically. This is an effective way to restrict cryptographic attacks, such as a key compromise. Furthermore, the lifetime of a public-key in a certificate may be different from that of other information in it. Namely, it is not a good solution to issue a currently existing certificate, such as X.509, that contains attribute information as well as public-key information. Even though using a bundled (public key and attributes) X.509 certificate increases performance (because simple mechanism and protocol are required to use both identity and attribute information on the Web), it does not support effective certificate management. Adding, deleting, or changing

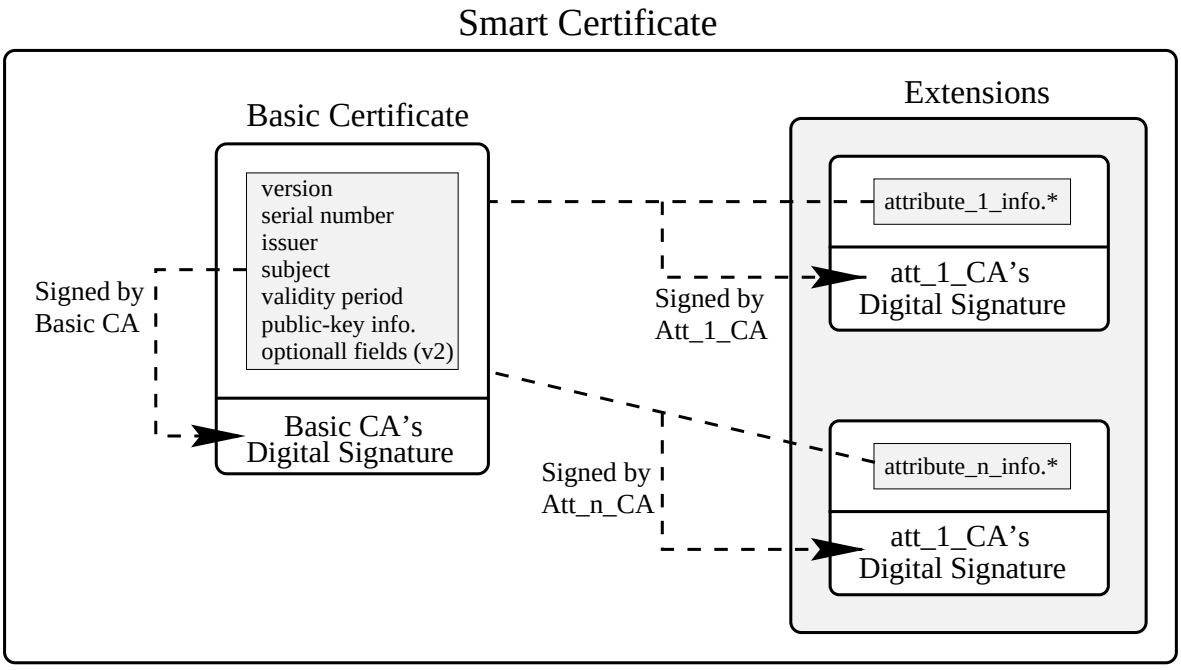


Figure 5.1: Attributes Signed by Multiple CAs in a Smart Certificate

attributes involves replacing and sometimes revoking X.509 certificates. This creates very large CRLs (Certificate Revocation Lists); therefore, it overburdens certificate management infrastructures. Furthermore, an organizational policy usually requires different authorities for maintaining attributes and public-keys. Since the current X.509 certificate cannot satisfy all the above requirements, we were motivated to design the *smart certificates*, described in the following subsections, to solve those problems. Note that a smart certificate is compatible with an X.509, since it keeps the same data format as an X.509.

If we use the extension fields in an X.509 certificate effectively, as depicted in Figure 5.1, we can separate the authority for attribute-issuing from the one for public-key-issuing. In other words, after a public-key authority (basic CA) issues an X.509 basic certificate for a user, Alice, an attribute authority (for instance, Att_1_CA) adds attributes for Alice to an extension field of the basic certificate (which contains public-

key information). Consequently, the attribute authority (Att_1_CA) signs on the basic certificate and the attributes he added, and puts the signature to another extension field in the basic certificate. This can happen multiple times on a basic certificate by different attribute authorities. Later, the identity verification should precede the attribute verification. For instance, another party, say Bob, verifies Alice's identity first by the basic CA's signature in the smart certificate. If the authentication is successful, Bob verifies Alice's attributes by the corresponding attribute authority's signature in the extension field. If the attributes are valid, then Bob uses those attributes for his purposes. The contents of the attribute information in a smart certificate depend on applications.

The public-key and the attributes can be maintained independently. For instance, even though Alice's attributes issued by her school-attribute authority are expired (revoked) in the certificate, the rest of the attributes, such as attributes issued by her company-attribute authority, and public-key information in her basic certificate, are still valid. Each attribute authority has independent control over the attributes he issued. For example, the school-attribute authority for Alice can change, revoke, or re-issue the school attributes in Alice's certificate. Intuitively, if her basic certificate is expired (revoked), then all the attributes become meaningless.

Furthermore, if we use the short-lived certificate mechanism, we do not need to worry about revoking the bundled certificates. As a result, a smart certificate supports high performance by bundling public-key information and attributes in a certificate without losing effective certificate management.

5.1.3 Support for Postdated and Renewable Certificates

Postdated[1] certificates are used to run a job to be performed at some time in the future. Suppose a security officer wants to issue a certificate for a user starting a

[1]Both postdated and renewable concepts are adopted from postdated and renewable tickets of Kerberos [SNS88, Neu94].

week from now and valid for 10 hours of use. He may issue a certificate with an expiration period of one week plus 10 hours from the present time. This is not an appropriate solution, since the certificate would be valid from the time it was issued until it expires. However, if we allow a certificate to become valid at some point in the future, we can satisfy the requirement.

Furthermore, if Alice needs a long-lived certificate, say, lasting for one year, it is more secure and efficient to issue a certificate that will be valid for that full 12 months, only if Alice keeps renewing it for a much shorter period, say, every day. To support this, we need to set the time (in the extension field), which specifies that the certificate cannot be renewed. Since the certificate is renewed every day, we do not need CRLs. If there is some reason to revoke Alice's certificate, the CA does not renew the certificates.

5.1.4 Encrypting Sensitive Information in Certificates

The smart certificates will support the encryption of some, or all, attributes, such as passwords, roles, or credit card numbers. Such an encrypted attribute in the certificate can be decrypted by an appropriate server using the corresponding key (the server's shared secret key or its private key).

5.2 Certificate Management

In this section, we want to compare smart certificates with existing certificates, such as X.509 and attribute certificates, in terms of the certificate lifetime and authentication of the owner of the attributes.

Usually, an X.509 certificate has a long lifetime, which requires an additional revocation mechanism (e.g., CRLs). Therefore, it has a relatively higher probability of being attacked, since the corresponding private key can be left in a system without the owner's knowledge, especially if the private key is stored in multiple machines. On the contrary, when we use a smart certificate, the short lifetime eliminates the

additional revocation mechanism (e.g., CRL for X.509), and makes the system more secure, since the remaining private keys expire shortly and automatically.

Currently, existing attribute certificates refer to another type of basic certificates, such as X.509, for authentication service. This mechanism brings complexities for the protocol itself and for certificate administration. For instance, an attribute certificate and the corresponding X.509 are issued in different entities and managed separately. Even the revocation mechanism is separate. The idea was brought to support separate authorities for attributes and authentication services.

However, if we use a smart certificate, both the attributes and public-key information can be bundled in a single certificate. This provides simplicity for both the protocol itself and for certificate administration. When we need separate authorities for attributes and authentication services, each authority signs separately on the same basic certificate and corresponding extension field, which contains attribute information. This can happen multiple times on a basic certificate by different attribute authorities. Each attribute authority has independent control over the attributes he issued. Even though a smart certificate can support independent management for the public key information and attributes, if there is one authority who controls both sets of information, the system management becomes simpler.

5.3 An Example of a Smart Certificate

Figure 5.2 shows an example of a smart certicate that a single issuer (certificate authority (CA)) issued for a subject, Alice. This smart certificate provides confidentiality service with an encrypted role information as an attribute in its extension field.[2] Only a server with the corresponding key can decrypt the encrypted information, and uses the role information for its purposes (in this example, for role-based access control). The smart certificate is postdated, since it becomes valid at some

[2]Since the smart certificate has both the subject's (Alice) identity and attribute (role), it inherently supports the user-pull model described in Chapter 3. Alice can obtain and present both her identity and attribute to the Web server using a single smart certificate.

Certificate Content:

```
Certificate:
    Data:
        Version: v3 (0x2)
        Serial Number: 26 (0x1a)
        Signature Algorithm: PKCS #1 MD5 With RSA Encryption
        Issuer: CN=data.list.gmu.edu, OU=LIST, O=GMU, C=US
        Validity:
            Not Before: Sun May 02 17:25:31 1999
            Not  After: Mon May 03 01:25:31 1999
        Subject: CN=Alice List, UID=alice, OU=LIST, O=GMU, C=US
        Subject Public Key Info:
            Algorithm: PKCS #1 RSA Encryption
            Public Key:
                Modulus:
                    00:9d:31:41:cf:45:d3:25:10:41:b3:ca:23:f6:09:91:ad:3d:
                    2d:c0:62:e1:ff:24:43:fe:39:90:c0:13:03:11:b5:77:ec:79:
                    17:b8:63:be:aa:36:4e:29:08:9b:76:64:b7:97:94:19:06:a7:
                    7a:b2:8b:31:f3:b5:72:3f:04:8f:17
                Public Exponent: 65537 (0x10001)
        Extensions:
            Identifier: Certificate Type
                Critical: no
                Certified Usage:
                    SSL Client
                    Secure E-mail
            Identifier: role
                Critical: no
                Value: hEwDNMBBleJQrWEBAgCS8TzT2/NMvn/xrkRsq/fRMSV3klUTEYkZoI
            Identifier: Authority Key Identifier
                Critical: no
                Key Identifier:
                    a5:d7:08:bc:ff:07:bd:5a:d4:8d:d4:68:53:87:4b:af:81:90:
                    f0:4d
    Signature:
        Algorithm: PKCS #1 MD5 With RSA Encryption
        Signature:
            c7:39:f7:b8:59:19:52:1c:fc:08:7c:11:f6:6e:5a:07:5b:55:80:a5:d8:
            65:a4:40:dc:06:5e:e4:ff:96:ad:71:9b:21:7a:4b:be:50:48:c2:f1:a6:
            7c:16:12:61:c7:bf:57:07:6d:c5:f4:f8:c2:e1:62:27:f6:d6:ae:09:77:
            46
```

Figure 5.2: An Example of a Smart Certificate

point in the future. Furthermore, it has a short-lived lifetime, which does not require the revocation scheme (CRL) for the certificate. In this example, the certificate was issued by only one CA, but it is also possible to be issued by multiple CAs, as we described in the previous subsection 5.1.2.

5.4 Applications of Smart Certificates

In this section, we introduce some applications of smart certificates. Many other applications can be similarly configured. Selection of the new features of smart certificates depends on applications and a given situation. Using the bundled (attributes and identifications) certificates is a good solution for the user-pull model, since the model requires the binding of this information. In contrast, it is not a good idea to use the bundled certificates for the server-pull model, since users do not need access to their attributes.

5.4.1 On-Duty Control

Suppose an employee, Alice, needs to receive a certificate at 9:00 a.m. every morning and use it until 5:00 p.m. Monday through Friday. This certificate contains her sensitive attributes, such as clearance or access control information; current X.509 cannot satisfy the requirements. It does not support the confidentiality service (by encryption) for some sensitive information in the certificate. If the validity period for the certificate is longer than that of Alice's on-duty hours, the privilege based on the certificate still remains even after her on-duty hours.

If we use the smart certificate, the above requirements can be securely satisfied. First of all, the sensitive information in the certificate is encrypted, providing the confidentiality service. By using the short-lived certificate feature, we can set the validity of the certificate only from 9:00 a.m. to 5:00 p.m. every day. Furthermore, the postdated-certificate feature allows the employee to receive a set of certificates on a certain day - for instance, five certificates for individual days (Monday through

Friday). Alice can then use the corresponding certificate each day during her on-duty hours. In this case, Alice does not need to receive the certificate every morning, and does not have the privilege based on the certificates after her on-duty hours. Alternatively, if we make the certificate valid for a set period of duration, for instance, from 9:00 a.m. to 5:00 p.m., but not between 5:00 p.m. and 9:00 a.m., the employee needs only one certificate for a week. However, this approach has a higher probability of compromise and may require using CRLs.

5.4.2 Attribute-Based Access Control

When a user, let's say Alice, using a Web browser contacts a Web server that has been configured to request smart certificates (which contain users' attributes), the browser is required to present Alice's smart certificate and prove that she is the rightful owner. After client authentication, the Web server makes access control decisions based on attributes (e.g., roles, clearance, and group membership) contained within the smart certificate itself.

5.4.3 Electronic Transactions

If a merchant site uses smart certificates (which contain customers' encrypted credit card numbers), customers do not need to key their credit card numbers in for every transaction. For more convenient service, the merchant can issue a smart certificate containing special tokens for customers, such as electronic coupons (which have the coupon's ID number and discount information). For instance, if Alice received an electronic coupon contained within her smart certificate, she can use it before the coupon's expiration date at the merchant site. In this case, the merchant site needs to keep a record of the coupon to protect replay usages of the same coupon.

5.4.4 Eliminating Single-Point Failure

Usually, a merchant site has a customer-information database maintained on a server. One of the disadvantages of this method is that if the server keeping customers'

information is penetrated by an attacker, all the customers' information, such as credit card numbers, preferences, addresses, and other sensitive information in the server, is open to the attacker. Furthermore, if a domain has multiple servers with multiple customer-information databases, maintenance and synchronization of this information is burdensome. There are also significant privacy concerns about data stored by servers, since such data can easily be misused. Users may feel more comfortable with servers that pledge not to maintain such data.

Smart certificates can solve these problems especially in the electronic commerce field. If a merchant site issues and uses smart certificates, the site does not need to have a customer-information database unless the site needs to track customers' access histories, since each customer's sensitive information is distributed and stored securely in the customer's smart certificates. Using smart certificates provides a more secure environment by eliminating customer-information databases, which can cause a single-point failure. Furthermore, the merchant can reduce the cost for the maintenance of customer-information databases.

5.4.5 Replace X.509

Besides using the extension fields in an X.509 certificate, a smart certificate provides new features; containing attributes, eliminating CRLs by short-lived certificates, providing multiple CAs, supporting postdated and renewable services, and encrypting sensitive information in the certificates. However, it is still compatible with X.509, since a smart certificate keeps the same data format as the X.509. For instance, as the original X.509 supports the Secure Socket Layer (SSL) protocol for secure communications between clients and servers, the smart certificates can also be used as server certificates and client certificates for SSL without modifying the protocol.

5.5 Summary

In this chapter, to support secure attrubute services on the Web, we have extended an existing digital certificate, X.509, with several new features. The extended certificates, *smart certificates*, provide a short-lived lifetime, attributes, multiple CAs, postdated and renewable services, and confidentiality services in the certificates. According to the requirements of applications, some of these new features can be selectively used in conjunction with currently existing technologies without changing standard protocols on the Web.

CHAPTER 6

IMPLEMENTATION

Current approaches to access control on Web servers are mostly based on user identities. A successful marriage of Web and RBAC (Role-Based Access Control [San98] technology can support effective enterprise-wide security in large-scale systems.

Figure 6.1 shows a schematic of RBAC on the Web. The role server has user-role assignment information for the domain. After a successful user authentication, the user receives his or her assigned roles in the domain from the role server. Later, when the user requests access to a Web server with the assigned roles in the domain, the Web server allows the user to execute transactions based on the user's roles instead of identity. The Web servers may have role hierarchies or constraints based on their policies. Administration of the role server can be performed in a decentralized manner by administrators on the Web [SP98].

However, the important question arises: how can the Web servers trust the role information presented by users? For instance, a malicious user may gain unauthorized access to the Web servers by using forged role information. Therefore, we must protect the role information from being forged by any possible attacks on the Web as well as in the end-systems.

There can be many possible ways to support the above requirement. In this chapter, we describe how to protect the role information from possible threats using *secure cookies* and *smart certificates*, and how to use the role information for RBAC1

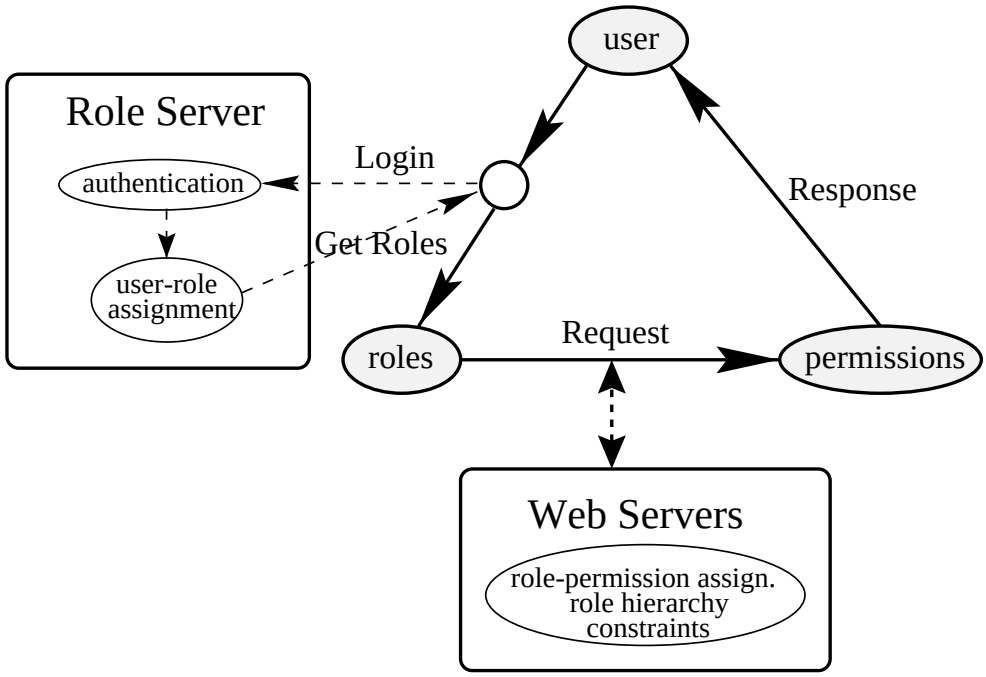


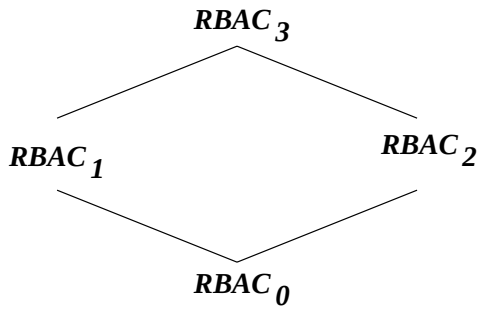
Figure 6.1: A Schematic of RBAC on the Web

(Role-Based Access Control with a role hierarchy) on the Web as one possible application of our new technologies.

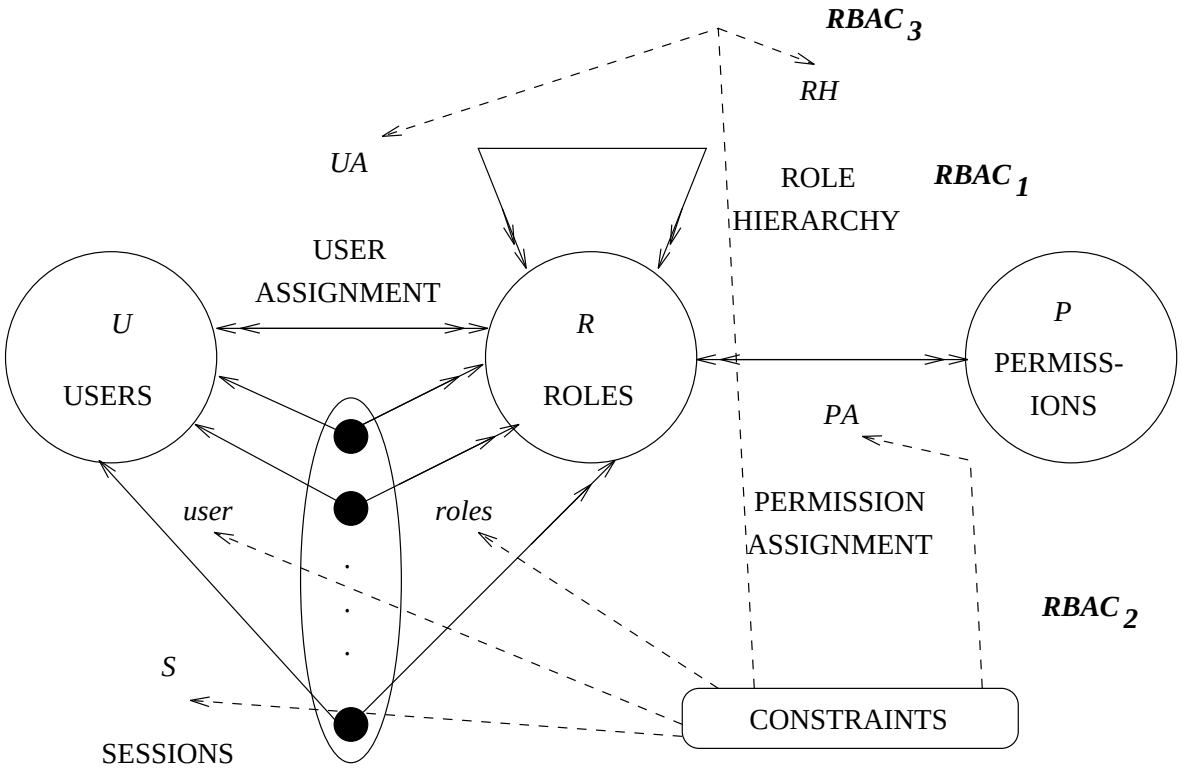
6.1 Role-Based Access Control (RBAC) Overview

Role-based access control (RBAC) has rapidly emerged in the 1990s as a promising technology for managing and enforcing security in large-scale enterprise-wide systems. The basic notion of RBAC is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies security management. A significant body of research on RBAC models and experimental implementations has developed [FK92, FCK95, Gui95, GI96, MD94, HDT95, NO95, SCFY96, vSvdM94, YCS97, San98]).

We were motivated by the need to manage and enforce the strong access control technology of RBAC in a large-scale Web environment, since RBAC is a successful



(a) Relationship among RBAC models



(b) RBAC models

Figure 6.2: A Family of RBAC Models

technology that will be a central component of emerging enterprise security infrastractures. Therefore, we have decided to implement the RBAC on the Web as one possible application of secure attribute services. Precisely, we have chosen RBAC1 (Role-Based Access Control Model with Role Hierarchy) among the RBAC families, because, we believe, role hierarchies are very useful to support the organization's policy.

Role-Based Access Control (RBAC) is a promising alternative to traditional discretionary and mandatory access controls, which ensures that only authorized users are given access to certain data or resources. It also supports three well-known security policies: data abstraction, least-privilege assignment, and separation of duties.

A role is a semantic construct forming the basis of access control policy. With RBAC, system administrators can create roles, grant permissions to those roles, and then assign users to the roles on the basis of their specific job responsibilities and policy. Therefore, role-permission relationships can be predefined, making it simple to assign users to the predefined roles. Without RBAC, it is difficult to determine what permissions have been authorized for what users.

Access control policy is embodied in RBAC components, such as user-role, role-permission, and role-role relationships. These RBAC components decide if a particular user is allowed access to a specific piece of system data.

Users are assigned to sessions during which they may activate a subset to which they belong. Each session can be assigned to many roles, but it maps only one user. The concept of a session corresponds to the traditional notion of subject in the access control literature.

Role hierarchy in RBAC is a natural way of organizing roles to reflect the organization's lines of authority and responsibility. In other words, junior roles appear at the bottom of the hierarchic role diagrams and senior roles appear at the top. The hierarchic diagrams are partial orders, which means they have reflexive, transitive, and an antisymmetric relation. An inheritance is reflexive because a role inherits

its own permissions to junior roles, transitive because of a natural requirement in this context, and antisymmetry rules out that inherit from one another and would therefore be redundant.

Constraints are an effective mechanism to establish higher-level organizational policy. They can apply to any relations and functions in an RBAC model. When applied, constraints are predicates that return a value of *acceptable* or *not acceptable*. A general family of RBAC models was defined by Sandhu et al. [SCFY96]. Figure 6.2 shows the most general model in this family.

RBAC0 is the base model that specifies the minimum requirement for any system that fully supports RBAC. RBAC1 and RBAC2 both include RBAC0, but they also have independent features. RBAC1 adds the concept of role hierarchies, which imply situations in which roles can inherit permissions from other roles. RBAC2 adds constraints that impose restrictions on components of RBAC. RBAC1 is incomparable with RBAC2, and vice versa. RBAC3 is the consolidated model that includes RBAC1 and RBAC2 and, by transitivity, RBAC0. The relationship between the four models of RBAC and the consolidated RBAC3 model is shown in Figure 6.2.

The bottom half of the figure shows administrative roles and administrative permissions, while the top half of the figure illustrates user roles and user administrative permissions in the system that determine control access to data and resources. There are five sets of entities in the RBAC96 model: users(U), roles(R), permissions(P), administrative roles(AR), and administrative permissions(AP). Regular permissions can only be assigned to regular roles while administrative permissions can only be assigned to administrative roles.

A user(U) is a human being or an autonomous agent; a role(R) is a job title or a job function in the organization with associated semantics concerning responsibility and authority; and a permission(P) is a description of the type of authorized interactions a subject can have with one or more objects. Administrative permissions control operations, which change the RBAC components, such as modifying the user and permission assignment and creating new roles and users in the system.

Details for motivation and discussion about the RBAC96 family of models, such as RBAC0, RBAC1, RBAC2, RBAC3, ARBAC0, ARBAC1, ARBAC2, and ARBAC3 are described in [SCFY96, San97].

6.2 RBAC1 on the Web by Secure Cookies

Cookies are used widely on the Internet today and they are becoming a standard as an integral part of Web technology. The cookie mechanism is inherently user-pull, that is, browsers receive cookies from Web servers, so there is no server-pull architecture alternative by secure cookies. Cookies can be easily forged and manipulated in their native form. We propose to use secure cookies, described in Chapter 4, to protect against tampering and misuse of the attributes in the cookies.

We believe it is possible to use secure cookies as an alternative for RBAC on the Web [PSG99]. The implementation is based on the user-pull model. We use IP_Cookies for the host-based mode, while using Pswd_Cookies for the user-based mode (described in Chapter 4. The selected snapshots from our implementation are attached in Appendix A.

6.2.1 Designing Secure Cookies for RBAC on the Web

Secure cookies provide three types of security services: authentication, integrity, and confidentiality services. Selection of the kinds and contents of secure cookies depends on applications and a given situation. However, at least one authentication cookie and the Seal_Cookie - which provides the integrity service to the cookies - must be used with other cookies to frame basic security services, regardless of applications.

Figure 6.3 shows a set of secure cookies that we will create and use for RBAC on the Web. The Name_Cookie contains the user's name (e.g., Alice), and the Role_Cookie holds the user's role information (e.g., Director). The Life_Cookie is used to hold the lifetime of the secure-cookie set in its Cookie_Value field and enables the Web server to check the integrity of the lifetime of the secure-cookie set.

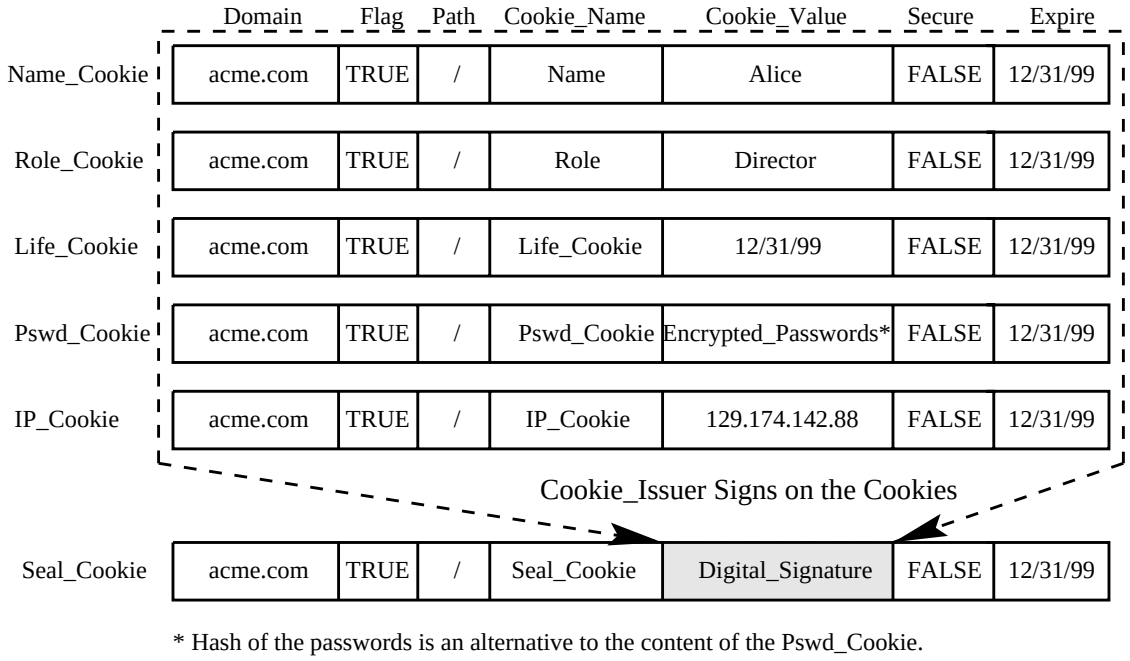


Figure 6.3: A Set of Secure Cookies for RBAC on the Web

To protect these cookies from possible attacks, we will use IP_Cookie, Pswd_Cookie, and Seal_Cookie. Authentication cookies (i.e., IP_Cookie and Pswd_Cookie) verify the owner of the cookies by comparing the authentication information in the cookies to those coming from the users. The IP_Cookie holds the IP number of the user's machine, and the Pswd_Cookie holds the user's encrypted passwords. This confidentiality service protects the values of the cookies from being revealed to unauthorized entity. In our implementation, we used the IP_Cookie and Pswd_Cookie together to show the feasibility, but only one of those authentication cookies can be used to provide the authentication service. The choice of an authentication cookie depends on the situation.[1] Finally, the Seal_Cookie - which has the digital signature of the cookie-issuing server on the secure cookie set - supports integrity service, protect-

[1] It is also possible for authentication to be based on use of RADIUS [RRSW97], Kerberos [SNS88, Neu94], and similar protocols. Our focus in this work is on techniques that make secure cookies self-sufficient rather than partly relying on other security protocols, which is always possible.

ing cookies against the threat that the contents of the cookies might be changed by unauthorized modification.

Figure 6.4 shows how the secure cookies (including a Role_Cookie) for RBAC are created and used on the Web. If a user, let's say Alice, wants to execute transactions in the Web servers in an RBAC-compliant domain, she first connects to the role server in the beginning of the session. After the role server authenticates Alice, it finds Alice's explicitly assigned roles in the URA (User-Role Assignment [SP98, SB97]) database and creates a set of secure cookies: Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie. Then, those secure cookies are sent to and stored in Alice's hard drive securely so that Alice does not need to go back to the role server to get her assigned roles until the cookies expire. Namely, she can use the roles in her Role_Cookie securely in the RBAC-compliant domain as long as the cookies are valid.

When Alice requests access to a Web server - which has PRA (Permission-Role Assignment [SBC+97]) information - by typing the server URL in her browser, the browser sends the corresponding set of secure cookies to the Web server: Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie. The Web server authenticates the owner of the cookies by using the IP_Cookie and Pswd_Cookie, comparing the value in the cookies with the values coming from the user. The user's passwords are encrypted in the Pswd_Cookie using the Web server's public key. The Web server decrypts the value of the Pswd_Cookie by using the corresponding key to read the user's passwords. Finally, the Web server checks the integrity of the cookies by verifying role server's digital signature in the Seal_Cookie using the role server's public key. If all the cookies are valid and verified successfully, the Web server trusts the role information in the Role_Cookie and uses it for RBAC with a role hierarchy and permission-role assignment information in the Web server.

There are basically two cryptographic technologies applicable for secure cookies: public-key-based and secret-key-based solutions. In our implementation, we use the public-key-based solution for security services provided by a PGP package via

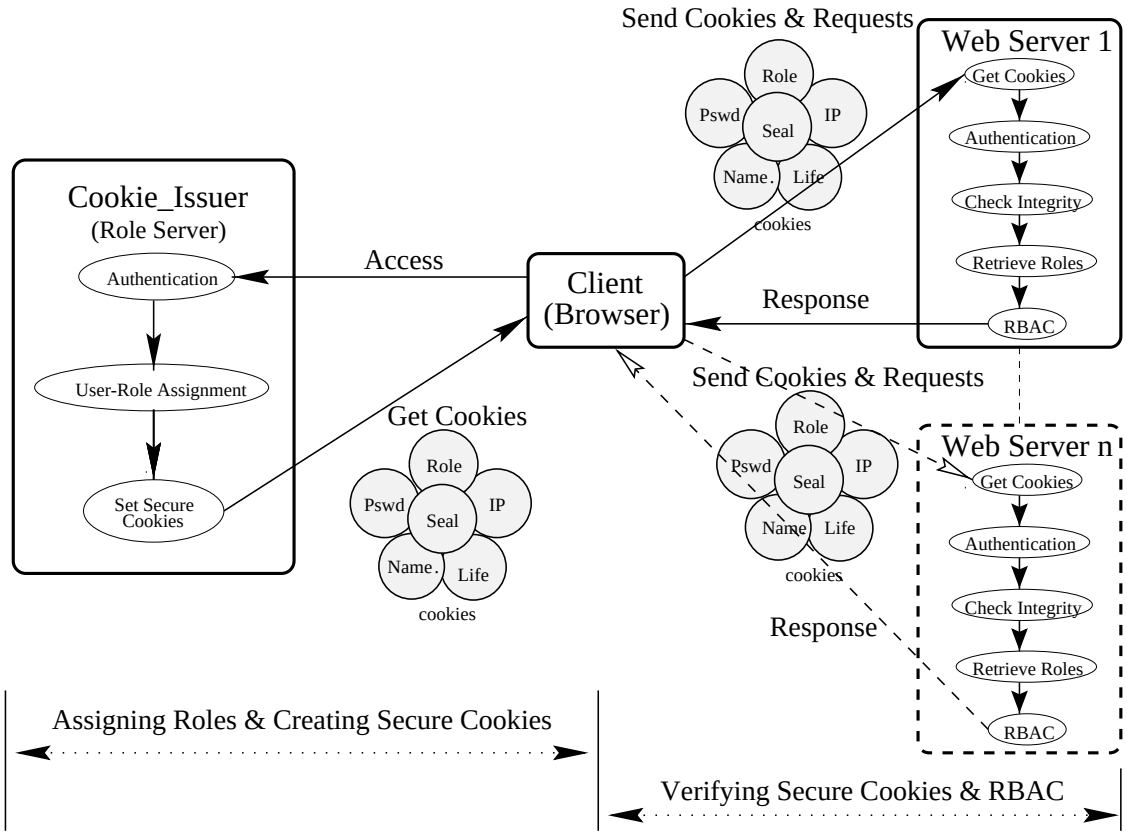


Figure 6.4: RBAC on the Web by Secure Cookies

CGI scripts [Her96]. In the following subsections, we will describe secure cookie creation, verification, and use of the role information in the Role_Cookie for RBAC with role hierarchies.

6.2.2 Secure Cookie Creation

When a user, Alice, connects to the role server (which supports HTTP) of the domain with her Web browser, she is prompted by the HTML form to type in her user ID and passwords for the domain. We used the POST[2] method to send the

[2]The GET request is very similar to the POST except that the values of the form variable are sent as part of the URL. However, the POST method sends the data after all their request headers

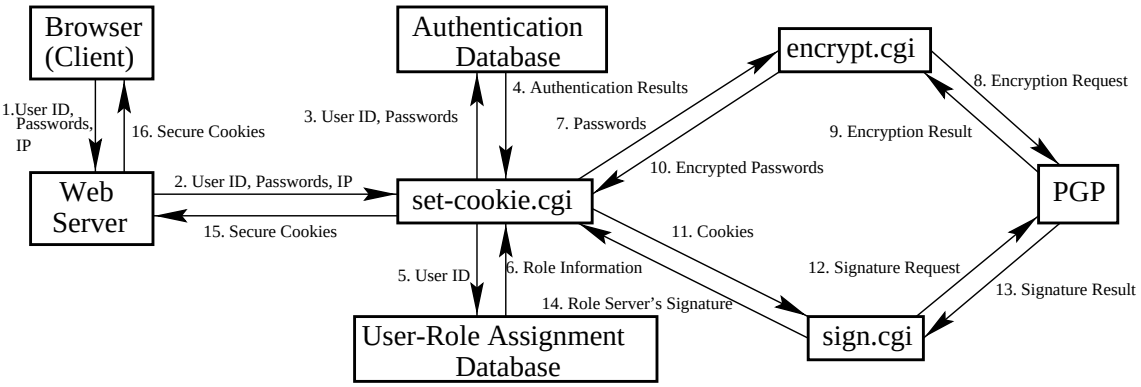


Figure 6.5: Creating Secure Cookies

information to the role server and the ACTION field to specify our Cookie-Set CGI program (set-cookie.cgi), to which the form data is passed. Figure 6.5 is a collaborational diagram in UML (Unified Modeling Language [BJR98]) style notation for secure cookie creation. This diagram shows how we create a set of secure cookies for our implementation (refer to the left side of Figure 6.4).

The Web server receives the request headers, which include the address to the "set-cookie.cgi" program on the server. The server translates the headers into environment variables and executes the program. The "set-cookie.cgi" program first retrieves the IP number of the client machine from the environment variable, REMOTE_ADDR, and the user ID and passwords using the read and split functions of Perl [Her96]. The program authenticates the user by comparing the user ID and passwords with the ones in the authentication database.[3] It then assigns the user to roles by matching the user ID and the corresponding roles from the URA (User-Role Assignment) database.

Subsequently, a subroutine for encryption is called to another CGI program (encrypt.cgi), which uses PGP to encrypt the passwords by the cookie-verifying Web

have been sent to the server.

[3]If the user already has an authentication cookie in a set of secure cookies, Web servers can use the authentication cookie for user authentication instead of authentication databases.

server's public key. These encrypted passwords will be stored in the Pswd_Cookie by the "set-cookie.cgi" program. Then, the "set-cookie.cgi" program creates IP_Cookie, Pswd_Cookie, Name_Cookie, Life_Cookie, and Role_Cookie, giving each cookie the corresponding value: IP number of the client machine, encrypted passwords, user's name, lifetime of the cookie set, and assigned roles.

To support the integrity service of the cookies, the "set-cookie.cgi" program creates a message digest of the cookies by MD5, and calls another CGI program (sign.cgi), which uses PGP to sign on the message digest with the role server's private key. The "set-cookie.cgi" then creates the Seal_Cookie, which includes the digital signature of the role server on the message digest of the cookies.

Finally, the Web server sends the HTTP response header, along with the cookies, back to the user's browser, and the cookies are stored in the browser until they expire. These secure cookies will be verified and used in the Web servers as described in the following subsections. Figure 6.6 is an actual snapshot of a set of secure cookies from our implementation that are stored in the user's machine after the cookies are generated by the cookie-issuing Web server. The contents of the cookies exactly reflect the ones presented in Figure 6.3. Each cookie has its corresponding domain, flag, path, security flag, expiration date, name, and value. The user's name, role, lifetime of the cookie set, IP number, encrypted passwords, and the digital signature of the cookie-issuing Web server on the cookies are stored in the corresponding cookies.

6.2.3 Secure Cookie Verification

Figure 6.7 is a collaborational diagram in UML style notation for secure cookie verification. This diagram shows how we verify (corresponding to the right side of Figure 6.4) the set of secure cookies that we generated in the previous subsection for our implementation. When Alice connects to a Web server (which accepts the secure cookies) in an RBAC-compliant domain, the connection is redirected to the "index.cgi" program. The related secure cookies are sent to the Web server and she

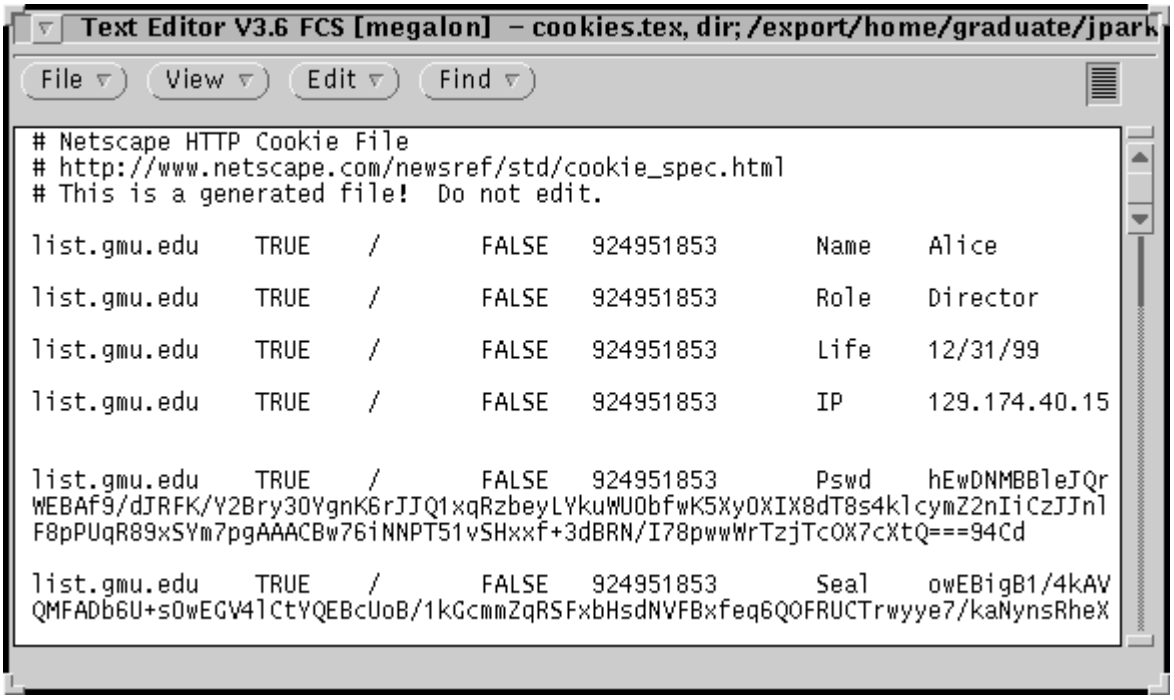


Figure 6.6: An Example of Secure Cookies Stored in a User's Machine

is prompted by the HTML form to type in her user ID and passwords. The "index.cgi" program first uses the HTTP_COOKIE environment variable to retrieve the secure cookies (Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, and Seal_Cookie) for the Web server. It then checks the validity of all the cookies. The two IP addresses, one from the IP cookie and the other from the environment variable, REMOTE_ADDR, are compared. If they are identical, then the host-based authentication is passed, and a hidden field "status" with the value of "IP-passed" is created to indicate that this stage was passed[4]. However, if the IP numbers are different, the user is rejected by the server.

[4]We used a hidden field to check the completion of the previous stage, which is passed on to the next program. This hidden field protects the pages from being accessed directly, skipping required verification steps, by a malicious user. For example, without this hidden field, a malicious user can access the pages directly with forged cookies.

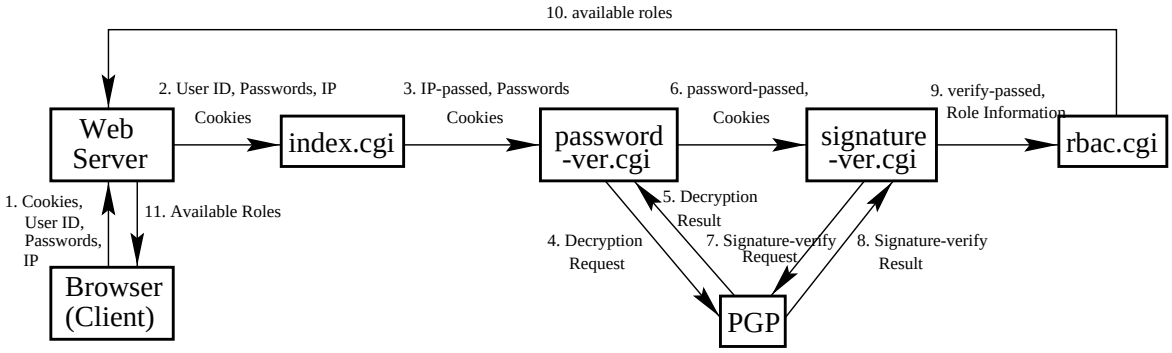


Figure 6.7: Verifying Secure Cookies

When the user submits her user ID and passwords to the server, the Web server translates the request headers into environment variables, and another CGI program, "password-ver.cgi," is executed. We used the POST method to send the information to the Web server and the ACTION field to specify the CGI program (password-ver.cgi) to which the form data are passed. The first thing the "password-ver.cgi" does is to check the hidden field "status" to see if the previous stage was successfully completed. If this is "IP-passed," the program decrypts the value of the Pswd_Cookie (encrypted user password) using the PGP with the Web server's private key, since it was encrypted with the Web server's public key by the role server. The program (password-ver.cgi) then compares the two passwords: one from the user and the other decrypted from the Pswd_Cookie. If they are identical, then the user-based authentication is passed, and a hidden field "status" with the value of "password-passed" is created to indicate that this stage was passed. However, if the two passwords are different, the user has to start again by either retyping the passwords or receiving new cookies from the role server.

After the password verification is completed, another CGI program, "signature-ver.cgi," is activated to check the integrity of the cookies. Like the other programs, it first checks the value of "status" passed on from the previous program, and it proceeds only if it shows the user has been through the password verification stage. If the

value is "password-passed," then the program creates a message digest from the set of secure cookies by MD5 and verifies the signature in the Seal_Cookie with the role server's public key using PGP. If the integrity is verified, it means that the cookies have not been altered, and a hidden field "status" with the value of "verify-passed" is created to indicate that this stage was passed and forwarded to the final program, "rbac.cgi." This program uses the role information in the Role_Cookie for role-based access control in the server as described in the following subsection. However, if the signature verification is failed, the user has to start again by receiving new cookies from the role server.

6.2.4 RBAC in the Web Server

After verifying all the secure cookies, the Web server allows the user, Alice, to execute transactions based on her roles, contained in the Role_Cookie, instead of her identity. In other words, the Web server does not care about the user's identity for authorization purposes. This resolves the scalability problem of the identity-based access control, which is being used mostly in existing Web servers. Furthermore, the Web server can also use a role hierarchy, which supports a natural means for structuring roles to reflect an organization's lines of authority and responsibility. Each Web server may have a role hierarchy defferent from that in other servers. In our implementation, we used a role hierarchy in the Web server, depicted in Figure 6.8. The location of RBAC-compliant Web servers is geographically free from that of the role server, since cookies can be issued by one Web server for use by others, regardless of their physical location.

If the "rbac.cgi" program in Figure 6.7 receives the value, "verify-passed," from the previous verification step, it means that the cookies have successfully passed all the verification stages, such as IP, passwords, and signature verification. Therefore, the Web server can trust the role information in the Role_Cookie, and uses it for role-based access control in the server.

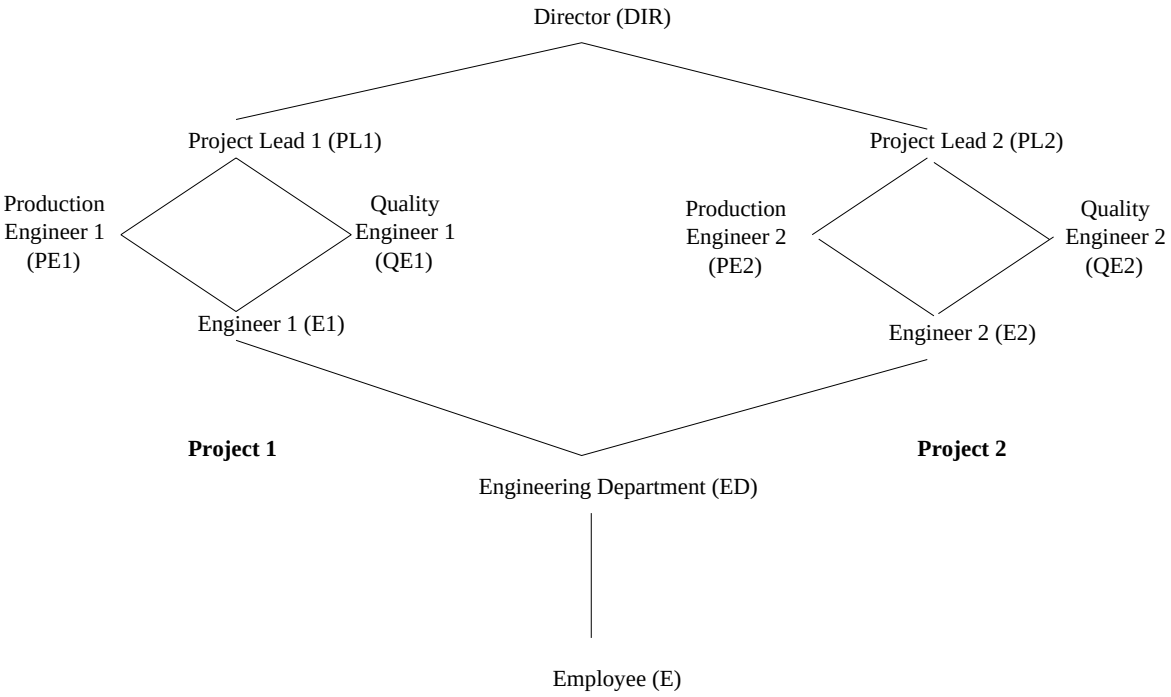


Figure 6.8: An Example of a Role Hierarchy

Suppose Alice has the role DIR[5] in her Role_Cookie and she wants to access resources for PE1 - which require the PE1 role or roles senior to the PE1 role in the role hierarchy - in a Web server. First, she has to prove that the cookies she is presenting are genuine. To prove this, she has to go through all the verification steps: IP, passwords, and signature verification. She cannot jump ahead or skip any verification stage, as each program requires a hidden field, "status," from the previous stage. After the Web server has successfully completed all the verification steps, the "rbac.cgi" program retrieves the role information, DIR, from Alice's Role_Cookie, and shows all the available roles[6] based on the role hierarchy depicted in Figure 6.8. Since she wants access to the pages that require PE1's privilege, she chooses the PE1 role

[5]Multiple roles can be stored in a Role_Cookie.

[6]In this example, all the roles from E to DIR are available to Alice, since she has the senior most role in the role hierarchy.

to activate it among her available roles. Then she has the permissions assigned to the PE1 role and the roles junior to PE1, such as E1, ED and E. Now, when Alice requests access to a particular page in the server, the page checks if her activated role, PE1, has permission to access the page.

The user can use any roles among her available roles by activating them. For instance, if Alice would activate PL1, then she would be allowed to access the pages, which require the PE1 role, since PL1 is senior to PE1 in the role hierarchy. However, if she were to activate E1, then she would not be allowed to access the pages, since E1 is junior to PE1. This supports least privileges, since only those permissions required for the tasks are turned on by activating the required role.

How then can the Web server protect the pages from being accessed by unauthorized users? Suppose a malicious user, Bob, has the role PE1 but wishes to access pages that require the PL1 role. He could change the value of his Role_Cookie so that it has PL1, or roles senior to PL1. He would go through the password verification stages, since he would be able to log in as Bob using his own passwords. However, when his Seal_Cookie is being verified, there would be a problem, as the signature verification would fail. Therefore, he would not be allowed to move beyond this stage. On the other hand, he could try accessing the pages directly by typing the URLs. This would not be allowed, since each page checks to see if he has activated the required role, PL1, or roles senior to PL1. In other words, Bob is not allowed to access the pages, which require roles senior to his, because he cannot activate the senior roles, which are out of his available role range.

As a result, the Web server allows only users, who have gone through all the verification steps with the secure cookies (Name_Cookie, Life_Cookie, Role_Cookies, IP_Cookie, Pswd_Cookie, Seal_Cookie), to access the pages. This access also is possible only if the users have the required roles and activate them among their available roles based on the role hierarchy.

6.2.5 Problems Faced and Solutions

In this section, we will describe some problems that we encountered during the implementation and the solutions that we found.

As we mentioned earlier, we used PGP to provide security services to the secure cookies. All the programs (PGP, CGI scripts, Web servers) are running under certain accounts in a UNIX environment. PGP is executed by the Web server via the proper CGI programs. However, PGP stores the private key in a key-ring file "secring.pgp," which is created when the key-pair is being generated by the cookie-issuer (or cookie-verifier), who owns the key-ring file. The permissions to this key-ring file have to be restricted to the owner of the key only for secure services. Therefore, PGP should be run as the owner of the corresponding key-ring files to access the keys. However, the CGI programs run as a default user account, usually *nobody*, with a minimum privilege in the Web server, hence the programs cannot execute PGP successfully as they do not have access to the corresponding key-ring files. Therefore, we had to find a way to make the CGI programs execute PGP with the permissions to access the corresponding key-ring files.

We decided to use *setuid*[7] to solve this problem. Because the owner of the PGP executive file is the same as the one for the corresponding key-ring files, we set the *setuid* bit on the PGP executive file. The CGI programs, such as encrypt.cgi, sign.cgi, password-ver.cgi, signature-ver.cgi, etc, are running as *nobody*. However, if they execute the PGP executive file, which has a *setuid* bit, then they are runnung the program with the owner's privilege so that they are able to access the key-ring files to finish the requested service.

Furthermore, whenever a user (the Web server in our implementation) needs to use PGP services, PGP requires a pass phrase to unlock its key holders. Since the Web server cannot type in the pass phrase in a UNIX shell as regular users do,

[7]The *setuid* file access mode in UNIX provides a way to grant users temporary access to which they are not otherwise allowed. After the process is completed, users do not have the access any more.

we had to find another way to solve this problem. Therefore, we used PGPPASS, an environment variable, to hold the pass phrase that PGP requires. This PGPPASS environment variable provides the pass phrase to the PGP whenever the Web server needs to access the PGP key rings.

Finally, we had to ensure that the users go through each stage of verification before viewing the final pages. To ensure this we used a hidden field "status" to check the completion of the previous verification step. Passing the variable from one program to the next, and verifying whether this variable has the right value before going on, ensures that no stage is skipped. The values for "status" vary in each verification step. For example the password-verifying program requires the value of "IP-passed," while the signature-verifying program requires "password-passed" from the previous stage.

6.2.6 Summary

In this section, we have described how we implemented RBAC with a role hierarchy on the Web using *secure cookies*. To protect the role information in the cookies, we provided security services, such as authentication, confidentiality, and integrity, to the cookies using PGP and CGI scripts in the Web servers. The cookie-issuing Web server creates a set of secure cookies including the user's role information, and other Web servers use the role information for RBAC with role hierarchies after cookie verification. This access control mechanism solves the scalibility problem of existing Web servers. The implementation is transparent to users and applicable to existing Web servers and browsers.

6.3 RBAC1 on the Web by Smart Certificates

In Chapter 5, we described how to render smart certificates by extending X.509 to support secure attributes services on the Web. In this section, we describe an implementation of Role-Based Access Control (RBAC) with role hierarchy on the

Web as one possible application of smart certificates [PS99a]. Even though smart certificates can support several new features, selection of the features depends on the types of applications that are being used. To support RBAC on the Web by this new technology, we issued a smart certificate - which holds the subject's role information - and configured a Web server to use the role information in the certificate for its access control mechanism. Since the subjects' role information is provided integrity, the Web server can trust the role information after certificate verification by SSL, and uses it for role-based access control. To keep the compatibility with existing technologies, such as SSL, without using an additional channel for attribute transfer on the Web, we used a bundled (subject's identity and attributes) smart certificate in the user-pull model (described in Chapter 3). In this implementation, we used a Netscape Certificate server and a Microsoft IIS 4.0 in Windows NT platform to support RBAC on the Web. However, this approach is also possible using other certificate servers or Web servers in different platforms. The selected snapshots from our implementation are attached in Appendix B.

6.3.1 Obtaining and Presenting Assigned Roles on the Web

Figure 6.9 shows how a bundled smart certificate is issued and used for RBAC on the Web. If a user, Alice, wants to execute transactions in the Web servers in an RBAC-compliant domain, she first connects to the role server in the beginning of the session. After the role server authenticates Alice, it finds her explicitly assigned roles in the URA (User-Role Assignment [SP98, SB97]) database and creates a smart certificate (which holds her explicitly assigned roles). Then, the smart certificate is sent to and stored in Alice's machine - which has Alice's private key corresponding to the smart certificate - so that Alice does not need to go back to the role server to obtain her assigned roles until the certificate expires. Consequently, she can use the roles in her smart certificate in the RBAC-compliant domain as far as the certificate is valid. In this implementation, we used the OU (Organization Unit) field in X.509 certificates to store each subject's role information.

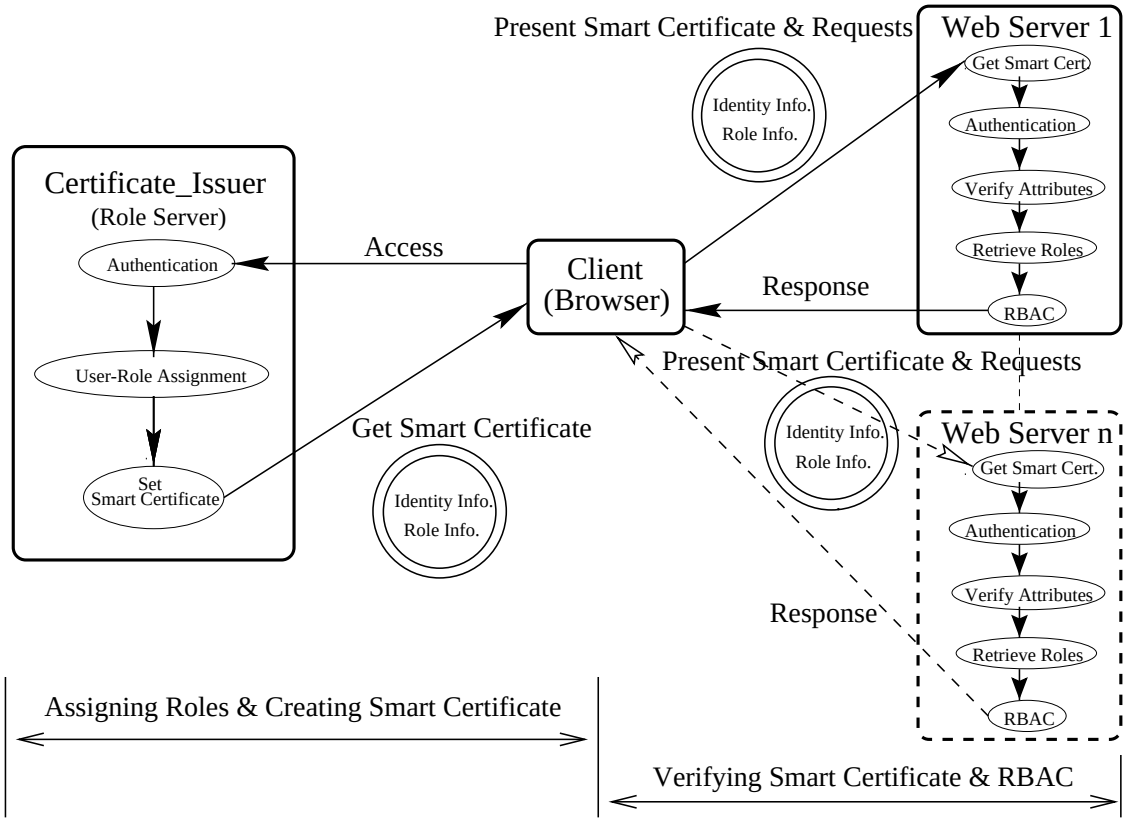


Figure 6.9: RBAC on the Web by Smart Certificate

When Alice requests access to a Web server - which requires clients' certificates and has PRA (Permission-Role Assignment [SBC+97]) information - by typing the server's URL in her browser, the browser and Web server authenticate each other over SSL. After the browser receives and verifies the server's X.509 certificate, it presents the client's smart certificate - which has Alice's role information - to the Web server. The Web server authenticates Alice by verifying the smart certificate. If the smart certificate is valid and verified successfully, the Web server trusts the role information in the certificate and uses it for RBAC with a role hierarchy and permission-role assignment information in the Web server, as described below.

6.3.2 RBAC in the Web Server

Internet Information Server (IIS) depends on Windows NT File System (NTFS) permissions for securing individual files and directories from unauthorized access. NTFS permissions can be precisely defined with regard to the users who can access the contents of the server and which permissions are allowed to the users, while Web server permissions are applied to all users accessing the Web server.[8] NTFS permissions apply only to a specific user or group of users with a valid Windows NT account.

In a Windows NT environment, we can control user access to the contents in a Web server by properly configuring the Windows NT file system and the security features of the Web server. When the user attempts to access the Web server, the server executes several access control processes to verify the user and determine the allowed level of access based on its policy.

To support RBAC with the role hierarchy depicted in Figure 6.8, we configured an IIS 4.0 with two creative ideas: *role accounts* and *PAA (Permission-Account Assignment)* in the Web server. These ideas are described in the following subsections.

[8]For instance, Web server permissions can control whether users visiting the Web site are allowed to view a particular page, run scripts, or upload information to the site.

6.3.2.1 Mapping Roles to Role Accounts

Since the Web server uses roles - denoted in the client smart-certificates - for its access control mechanism, regular user accounts are not necessary in the server.[9] Instead, we created the role accounts (e.g., Director, Project_Lead1, Project_Lead2, Project_Engineer1, Quality_Engineer1, and so on) in the Windows NT server, where the Web server (IIS 4.0) is installed. Each role account corresponds to a role in the role hierarchy in Figure 6.8. Then, by configuring the Web server's certificate mapping feature, we mapped each role in the role hierarchy to the corresponding role account in the Windows NT server. For example, we mapped the role DIR to the role account Director in the server. If a smart certificate has multiple roles in it, then it is mapped to multiple role accounts in the server. After a user (subject), Alice, authenticates to a Web server over SSL by sending her client smart-certificate - which has the role "DIR" - to the server, she is mapped to the role account "Director" in the Windows NT server. As a result, even though Alice does not have an account in the server, she acquires the Director's permission in the server, since she is assigned to the role "Director" - which is denoted in her smart certificate. The permission of each role account depends on the policy of the Web server.

6.3.2.2 Providing Role Hierarchy

How then can the Web server support the role hierarchy? Figure 6.10 shows how we used a built-in access control mechanism in the Windows NT server to support the role hierarchy depicted in Figure 6.8. Reflecting the roles in the hierarchy, we created the role accounts, such as Director, Project_Lead1, Project_Lead2, Project_Engineer1, Quality_Engineer1, and others. We also created directories in the Windows NT file system, where each directory has files to be accessed by a specific role in the role hierarchy. Subsequently, we configured the Windows NT file system to assign each role account to specific access rights to the directories based on the role hierarchy. For instance, the role account Project_Lead1 is assigned to access rights to the

[9] However, the Web server may need administrator accounts for its maintenance.

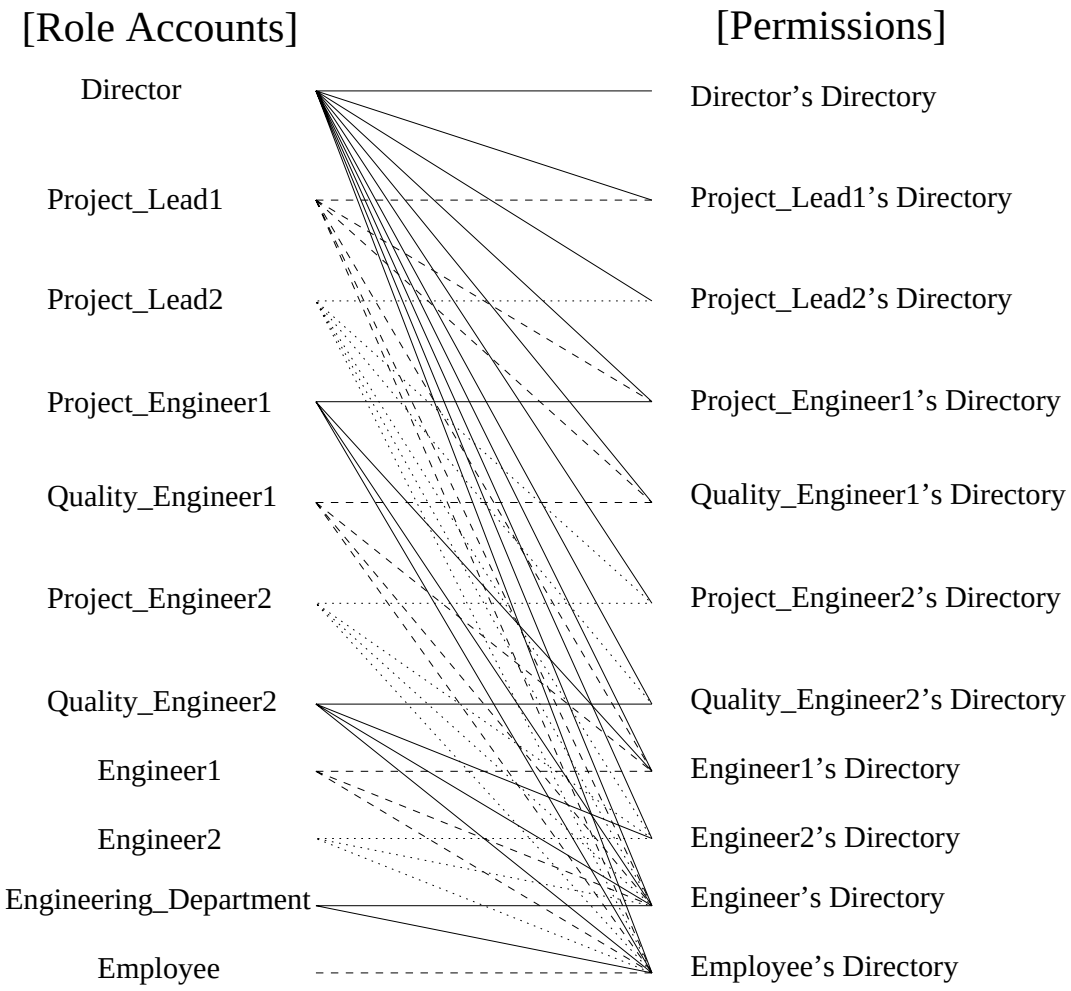


Figure 6.10: Role Accounts and Permission Assignment

Project_Lead1's directory - which has resources for the role Project_Lead1 - and the directories that require the roles junior to the Project_Lead1 role in the role hierarchy. In other words, if Alice is mapped to the role account Project_Lead1, she obtains permissions assigned to the role account Project_Lead1, thereby acquiring access rights to the directories for Project_Lead1, Project_Engineer1, Quality_Engineer1, Engineer1, Engineering Department, and Employee.

As a result, after verifying the smart certificate, the Web server allows the user, Alice, to execute transactions based on her roles - contained in the OU field of the certificate - instead of her identity. In other words, the Web server does not care about the user's identity. This resolves the scalibility problem of the identity-based access control, which is being used primarily in existing Web servers. Furthermore, since the Web server also uses a role hierarchy, it supports a natural means for structuring roles to reflect an organization's lines of authority and responsibility. Each Web server may have a role hierarchy defferent from that in other servers. The location of RBAC-compliant Web servers is geographically free from that of the role server, since smart certificates (which include the subjects' role information) can be issued by one certificate server for use by other Web servers, regardless of their physical location.

6.3.3 Summary

In this section, we have described how we implemented RBAC with role hierarchies on the Web using *smart certificates*. The certificate authority issues a smart certificate, including a subject's identity and role information, and Web servers use the role information for RBAC with role hierarchies after identity and attribute verification. This access control mechanism solves the scalibility problem of existing Web servers. The implementation is transparent to users and applicable to existing Web servers and browsers. In our implementation, we used the OU (Organization Unit) field in X.509 certificates to simply add subjects' roles, and both identity and roles are signed by a single certificate authority. However, if a smart certificate has different attributes (which need to be signed by different CAs), or obtains detailed attribute information,

such as validity for each attribute or attribute issuer, we need to use the extension fields of X.509 as described in Chapter 5.

6.4 Discussion

For secure attribute services on the Web, we may consider using other existing technologies, such as SHTTP (Secure HTTP [RS98, SR98]), and SSL (Secure Socket Layer [WS96, DA99]). However, none of these can prevent end-system threats to the attribute information. For example, once Alice receives some attributes from a Web server over a secure channel, she can change the contents of the cookies or give them to other people. Then Alice, or the person impersonating Alice, can access the Web server (which accepts the attributes) over another secured channel using the forged information. On the other hand, secure cookies and smart certificates protect attributes from network and end-system threats.

In our secure-cookie implementation, individual files with CGI scripts check to see if the user has the required role to access the files. This is a per-file access control mechanism. The per-file access control mechanism does not require the modification of the Web server; it works independently from the access control mechanism of Web servers. However, each HTML page needs to be written with a small CGI script to determine whether the user has the required role to access the file. Therefore, this mechanism is not limited to the location of the files. Even if a file moves to another directory or even other servers, the file still requires the corresponding role to access the file. Alternatively, we also used a separate CGI program (fetch.pl) to fetch the corresponding HTML files based on the user's roles. The Web page URL is not shown to the user, since the CGI program (fetch.pl) reads from the page and outputs it to the user. In this case, individual HTML pages do not need CGI scripts to check the user's roles. However, the access control mechanism is not free from the location of the files.

In contrast, we used the per-directory access control mechanism in our smart-

certificate implementation. We used a built-in ACL (Access Control List) in a Windows NT server to limit access to all the files in a directory to users, who have the corresponding role. To support RBAC, we need to configure the Web server to recognize the role information for its access control, while we do not need to change the HTML files in the server. However, if a file moves to other directories or servers, then the file is free from the access control, which affected on it in the previous directory. This also can be implemented in conjunction with the access control mechanism of other existing Web servers.

6.5 Related Work

6.5.1 *getAccess*

enCommerce has released *getAccess* [enC98] to implement a hierarchical role-based model for the organization online. Each role defines a specific access privilege to one or more resources. The roles can be grouped into *macro roles*, and macro roles can also have other macro roles. There are four main software modules in this product: registry server, access server, administration application, and integration tools. The access server is located in a company's Intranet or Extranet, while the registry server is always located in the Intranet. A user always connects to the access server first via browsers. The access server then connects the registry server to obtain the user's identification and roles through a secure connection. Subsequently, the registry server authenticates the user and returns the user's encrypted role information through cookies. These cookies are temporarily stored in RAM on the user's machine while the browser is open. When the user connects to a Web server in the Intranet, the browser sends the cookies to the Web server. The Web server decrypts and uses the encrypted role information in the cookies for role-based access control in the server.

The *getAccess* mechanism uses encrypted cookies. However, there is a huge difference between its approach and our secure cookies (described in Chapter 4). The encrypted cookies are not stored in the user's machine after the session. In other words, if a session is ended by closing the browser, the encrypted cookies disappear.

This means that whenever a user, Alice, needs to connect to a Web server with her roles, she must connect to the registry server first through the access server. On the contrary, secure cookies - which obtain the user's role information - can be stored in the user's machine after the session, even when the power of the user's machine is off. This is possible because the secure cookies can be provided integrity and authentication services as well as encryption. Therefore, once Alice obtains her secure cookies, she can use her roles until the cookies expire, without having to connect the cookie issuer.

6.5.2 *TrustedWeb*

Siemens Nixdorf released *TrustedWeb* [Nix98], which supports role-based access control for Web contents and applications, as well as security services, such as mutual authentication, integrity, and confidentiality for Intranets. The system, combining elements from both Sieman's SESAME [PP95] and Kerberos [Neu94], provides a single list of users on its central domain security server and assigns roles to the users. Therefore, access to the individual Web servers in the Intranet is controlled based on the role rather than the identity of the user. However, to use *TrustedWeb*, the client's browser needs specific software installed in the client's machine to communicate with the *TrustedWeb* servers in the Intranet while our techniques do not require any specific software in the client side.

6.5.3 *hyperDRIVE*

The *hyperDRIVE* [Bar97], which was developed by the Internal Revenue Service, is programmed in Java to provide a consistent, integrated, auditable, manageable RBAC implementation, employed by multiple servers, clients, and applications. The implementation includes a Java-capable and SSL-enabled Web browser, an LDAP[10] server, an Object Request Broker, and the *hyperDRIVE* client Java applet. *hyper-*

[10]Lightweight Directory Access Protocol [YHK95] defines a relatively simple protocol for updating and searching directories running over TCP/IP.

DRIVE makes it possible for servers, applications, and active objects with capabilities and facilities to consult the LDAP-hosted RBAC data. Through this consultation, the client's requests are accepted or rejected to support secure communication between clients and servers based on RBAC. The access includes location, methods, and modes of access to and of network computing resources. The authorization involves the mapping of one or more operations or privileges to a defined role.

However, it is difficult for *hyperDRIVE* to set constraints and role hierarchy, and administer many roles in a huge system, because its approach focuses on the user's transaction.

6.5.4 *I-RBAC*

The *I-RBAC* [TC97] implements RBAC with agents: coordination agents, task agents, and database agents. The agents are active network objects that implement the different security procedures by checking the user's authorizations for using resources within an Intranet. The main advantage of this approach is the existence of local and global network objects in the system. A local object has an identity known only with the corresponding local server, while a global network object has a unique identity known throughout the Internet. Therefore, when a network object with a given role wants to access Intranet resources, the system checks its role in the global-role database. If the object's global role exists in the global-role database, then the system derives its global permissions and uses the permission domain tables in the local-role databases, which contain the access information of the local server's network objects within an Intranet, to verify the global role's derived local authorizations.

However, there is a consistency problem between roles. For instance, if we have hundreds of roles and thousands of permissions in the system, it is very difficult to keep consistent by reflecting new security requirements between global network objects and local network objects.

CHAPTER 7

CONCLUSIONS

7.1 Secure Cookies vs. Smart Certificates

In this section, we compare secure cookies with smart certificates in user-pull and server-pull models.

Secure cookies inherently support only the user-pull model, since cookies are stored in users' machines; they cannot operate in the server-pull model. In contrast, smart certificates support both user-pull and server-pull models. A bundled (identity and attributes) smart certificate is useful for the user-pull model.

Both secure cookies and smart certificates are compatible with existing technologies; HTTP can support the secure cookie mechanism as it does for regular cookies, and SSL can support smart certificates as it does for X.509 certificates. However, for the server-pull model utilized by smart certificates, an additional channel is required for attribute transfer between the attribute server and Web servers.

To use secure cookies or smart certificates in the user-pull model, each user needs to obtain his or her attributes from the attribute server. Furthermore, it is non-trivial to update or revoke users' attributes and make it effective instantly in the user-pull model. For instance, if the user already pulled her attributes, the updated version in the attribute server would not become effective instantly. Namely, an additional synchronization process is required.

The secure cookie mechanism is transparent to users. Once a user, Alice, ob-

tains her attributes in secure cookies, she can use them to access other Web servers in the domain, specified in the cookies, as long as the attributes are valid. When the user connects to a Web server, the relevant secure cookies are selected and presented to the server by the browser, and expired cookies are deleted from the user's machine automatically. By this feature, secure cookies support users' convenience for maintaining and using their attributes frequently in diverse Web sites (for instance, pay-per-access).

To use smart certificates, user cooperation is required. Whenever the user connects to a Web server, which requires a smart certificate from the client, the user needs to select a proper certificate among her available certificates, and present it to the server. Once Web servers install a CA (Certificate Authority) certificate as an acceptable certificate under a certain policy, a client certificate can be used in many Web servers (even in different domains). For instance, Alice's smart certificate - which has her credit card information - can be used in many Web sites in different domains for electronic commerce on the Web.

Using smart certificates in the server-pull model does not require user cooperation to obtain the attributes. Instead, it requires each Web server to obtain each user's attributes. In this case, the users' attributes can be updated or revoked, and become effective instantly, because all the attributes are stored in the attribute server and pulled by the Web servers on demand. By this feature, use of smart certificates in the server-pull model is a good solution for the applications, especially, where dynamic attribute update is critical, such as stop-payment in electronic transactions.

7.2 Contributions

In this dissertation, we have identified the user-pull and server-pull models for secure attribute services on the Web. In the user-pull model, a user pulls appropriate attributes from the attribute server and then presents them to the Web servers. In the server-pull model, each Web server pulls appropriate attributes from the attribute

server as needed. Each model can be made to work, and we have provided an analysis of their relative advantages and disadvantages.

We also have developed secure cookies and smart certificates to support these models on the Web. Secure cookies are constructed by cryptographic technologies to support authentication, integrity, and confidentiality services. Authentication services verify the owner of the cookies. Integrity services protect against threats that the contents of the cookies might be changed by unauthorized modification. Finally, confidentiality services protect against the values of the cookies being revealed to an unauthorized entity. Smart certificates have new features, but they are still compatible with X.509 certificates. They are able to support short-lived lifetime and multiple CAs without losing effective maintenance, contain attributes, provide postdated and renewable certificates, and provide confidentiality. Selection of these new techniques for both secure cookies and smart certificates depends on applications.

Finally, to provide concrete examples of secure attribute services on the Web by our new technologies, we have used secure cookies and smart certificates separately to implement role-based access control with a role hierarchy on the Web. To provide varied experiences, secure cookies were implemented in a UNIX environment, while smart certificates were implemented in a Windows NT environment. Even though we have implemented only RBAC on the Web by our new technologies in this dissertation, we believe it is sufficient to show the feasibility that these technologies can be used for other applications on the Web.

7.3 Future Research

Based on the research work in this dissertation, we propose the following research areas, discussed below.

7.3.1 Binding Identity and Attributes

Attributes must belong to the owner of the attributes. For instance, Alice should not be allowed to use Bob's attributes, and vice versa, in distributed systems. How then can we bind attributes to the appropriate identity? In our implementation, we used a single digital signature on both identity and attribute information to support integrity service. We also introduced the individual digital signatures as a binder between identity and attributes in a smart certificate. However, we believe there are many possible ways to bind both kinds of information. For example, we can use a subject's name, identifier, public key, encrypted passwords, hash of the passwords, and serial number. We would need to analyze each binder and compare their relative advantages and disadvantages.

7.3.2 Implementation Issues

In this dissertation, we have introduced many possible applications of our new technologies (secure cookies and smart certificates), and implemented role-based access control with a role hierarchy on the Web as one possible application. In the future, we plan to implement more applications using our new techniques .

APPENDIX A

SNAPSHOTS FROM RBAC ON THE WEB BY SECURE COOKIES

In this appendix, we attach the snapshots of selected images from our implementation of RBAC on the Web by secure cookies. The implementation is based on the role hierarchy depicted in Figure 6.8. In this example, Alice first connects to the cookie-issuing role server to receive her assigned roles. The role server issues a set of secure cookies, including Alice's role information. Finally, the secure cookies are sent to and stored in Alice's hard drive securely, so that she can use them in other Web servers. Detailed procedures are described in Section 6.2.

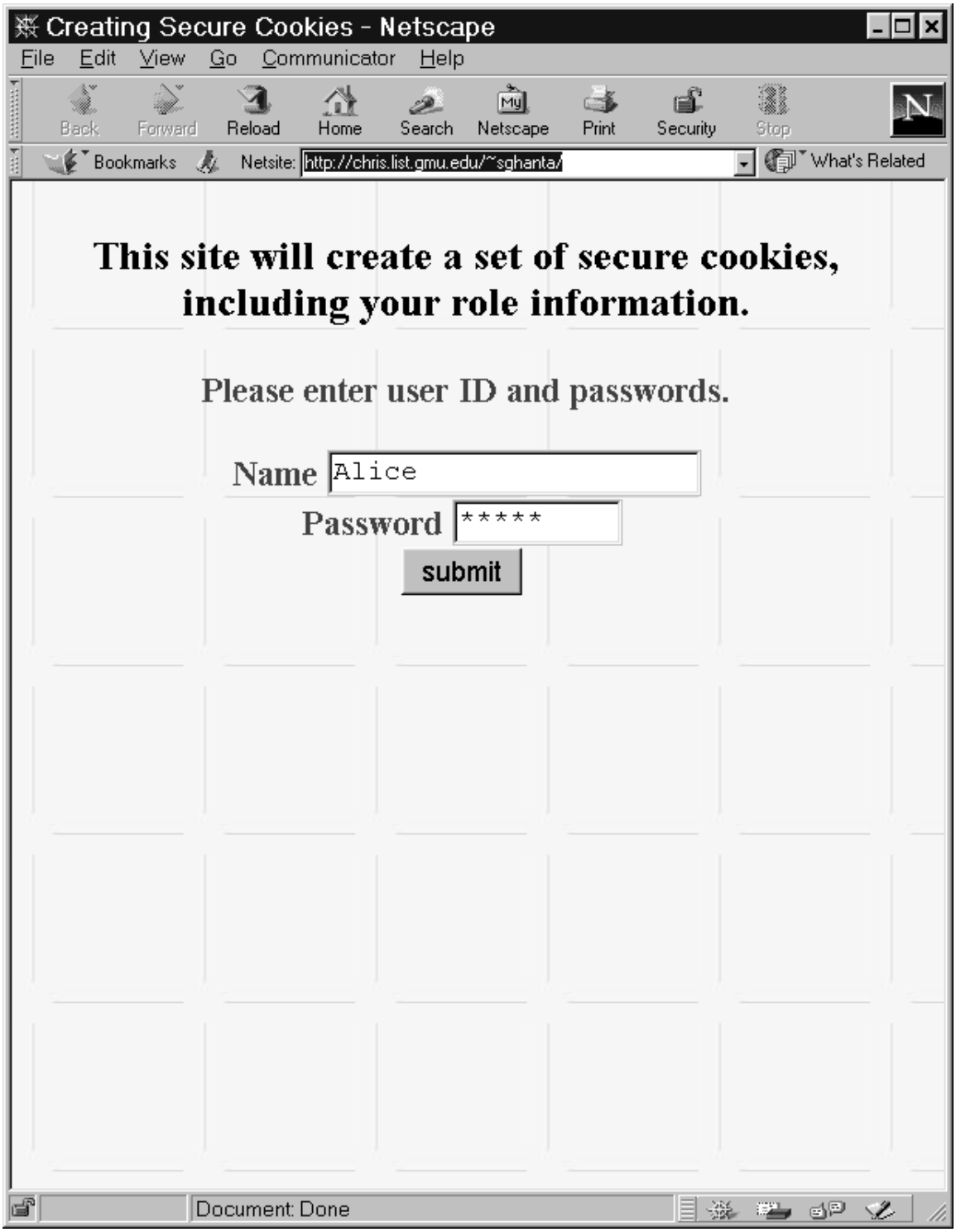


Figure A.1: Alice Connects to the Role Server

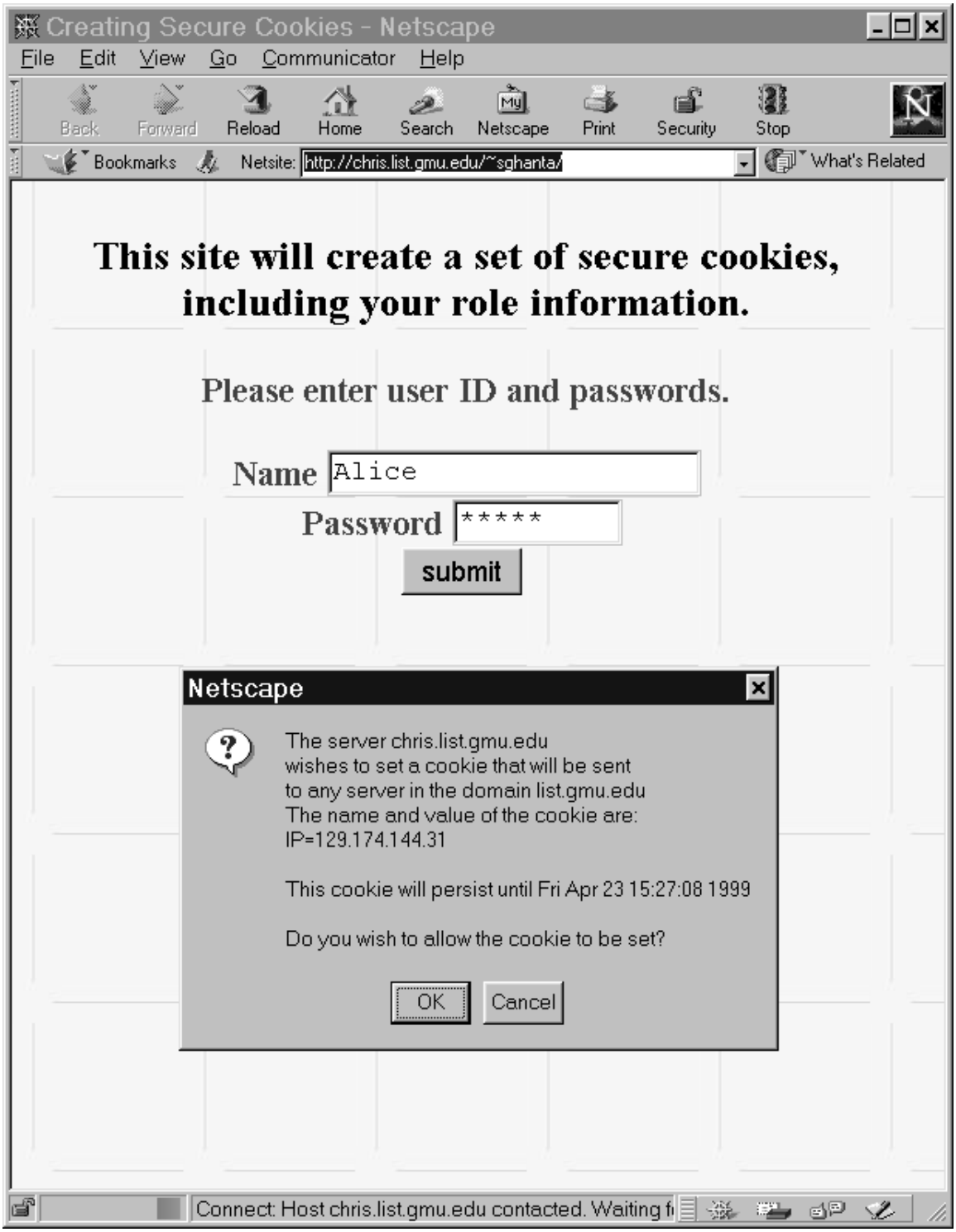


Figure A.2: The Role Server Issues an IP_Cookie for Alice

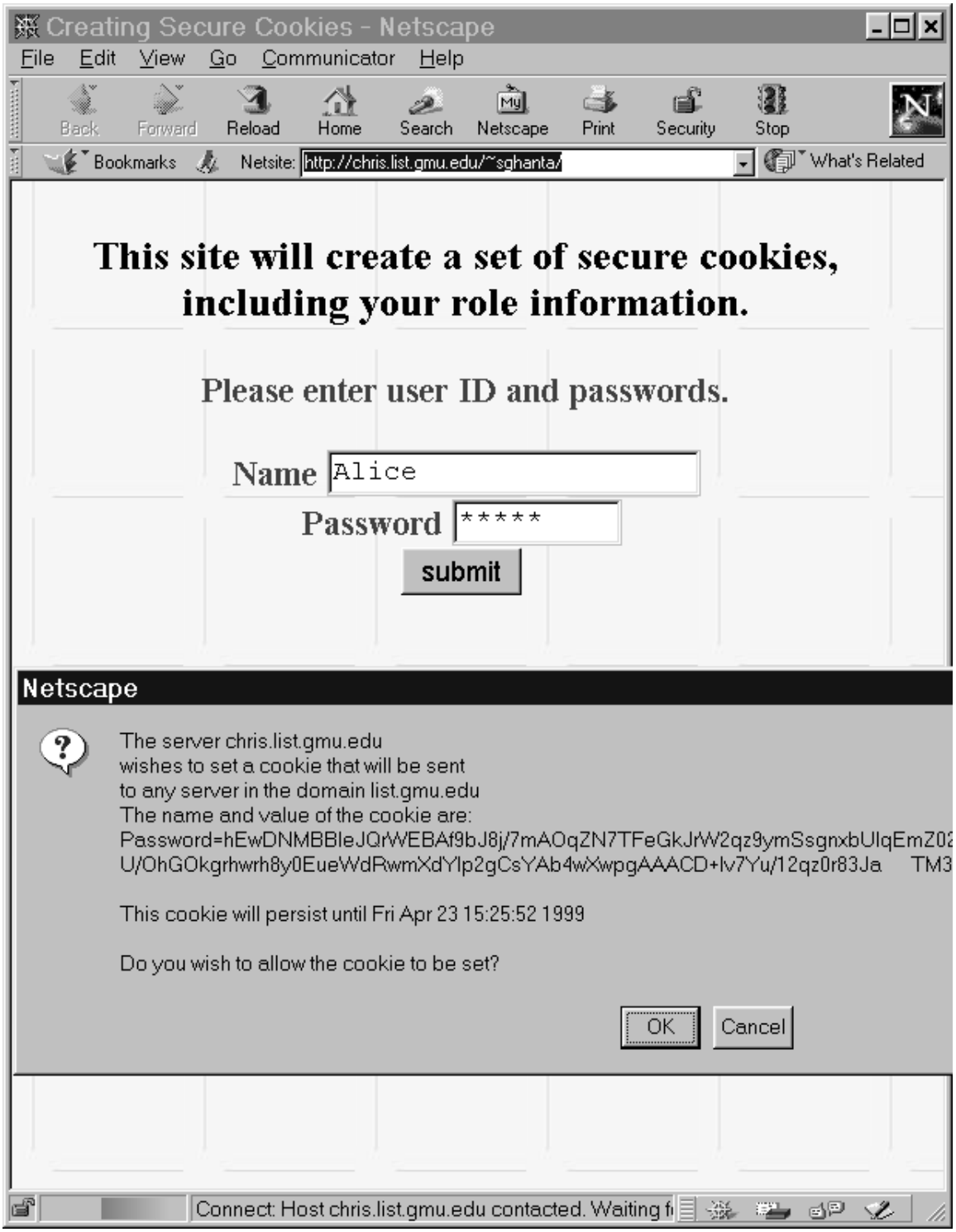


Figure A.3: The Role Server Issues a Pswd_Cookie for Alice

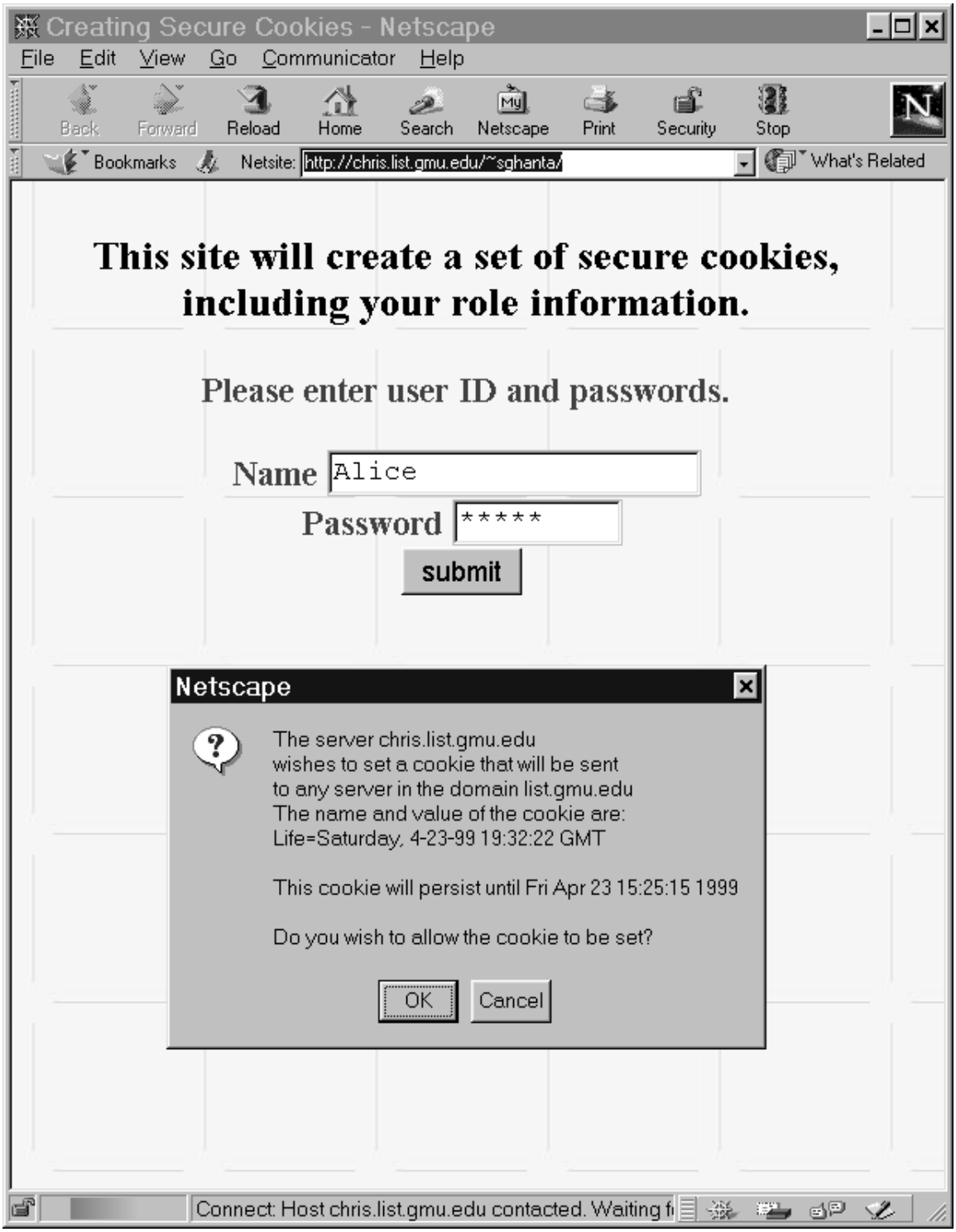


Figure A.4: The Role Server Issues a Life_Cookie for Alice

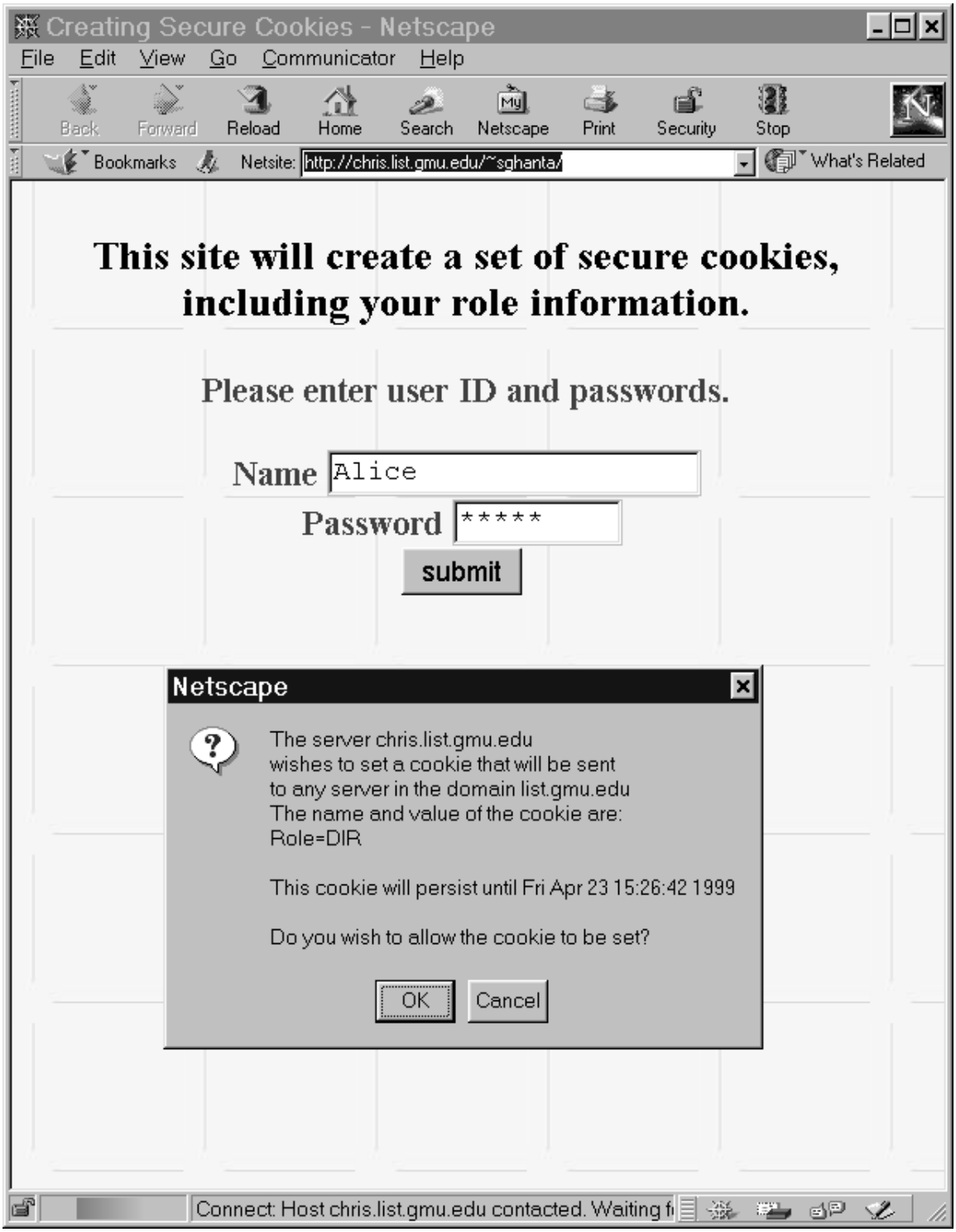


Figure A.5: The Role Server Issues a Role_Cookie for Alice

Figure A.6: The Role Server Issues a Seal_Cookie for Alice

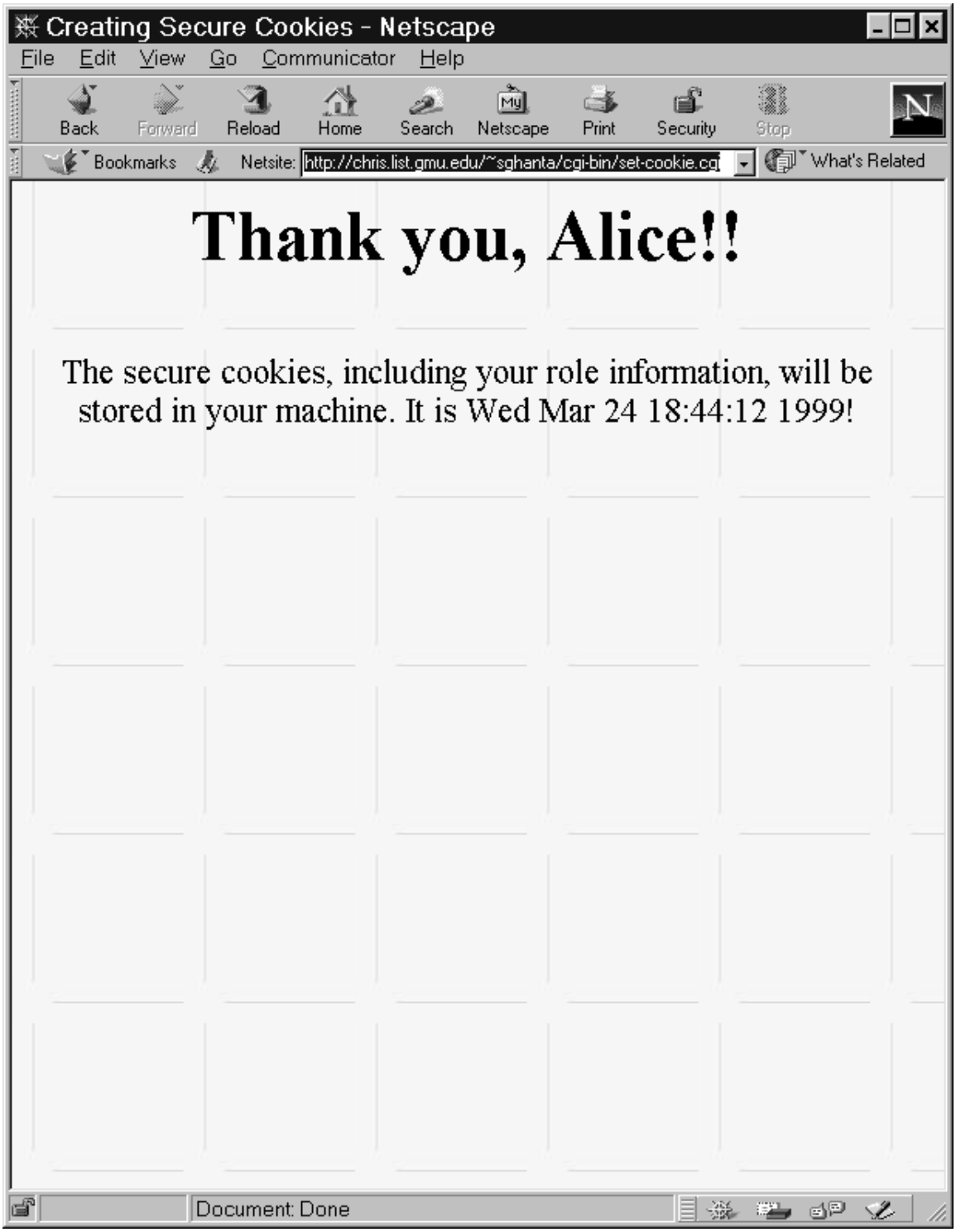


Figure A.7: Alice Stores Her Secure-Cookie Set

Figure A.8: Alice Connects to a Web Server

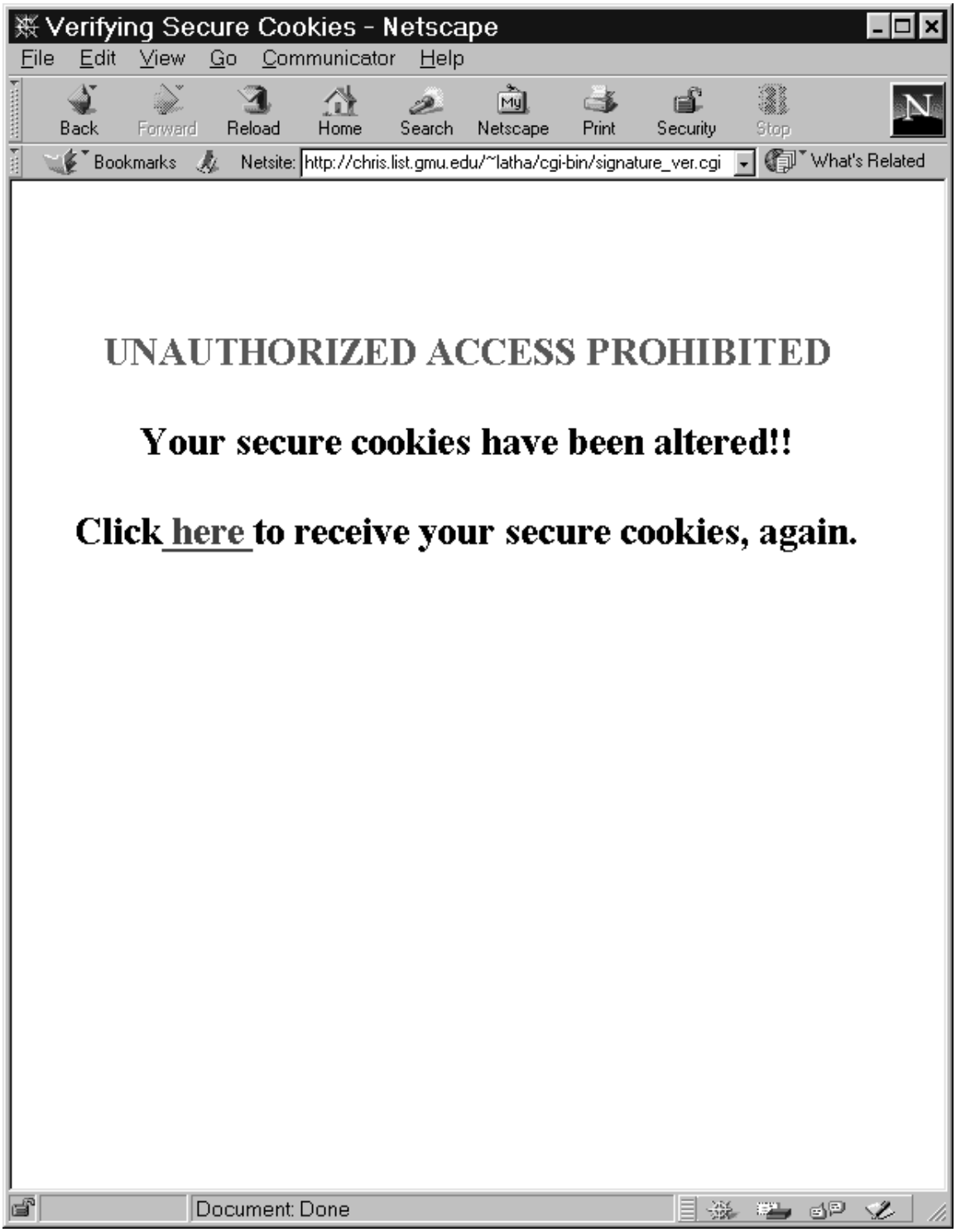


Figure A.9: An Example of Verification Failure

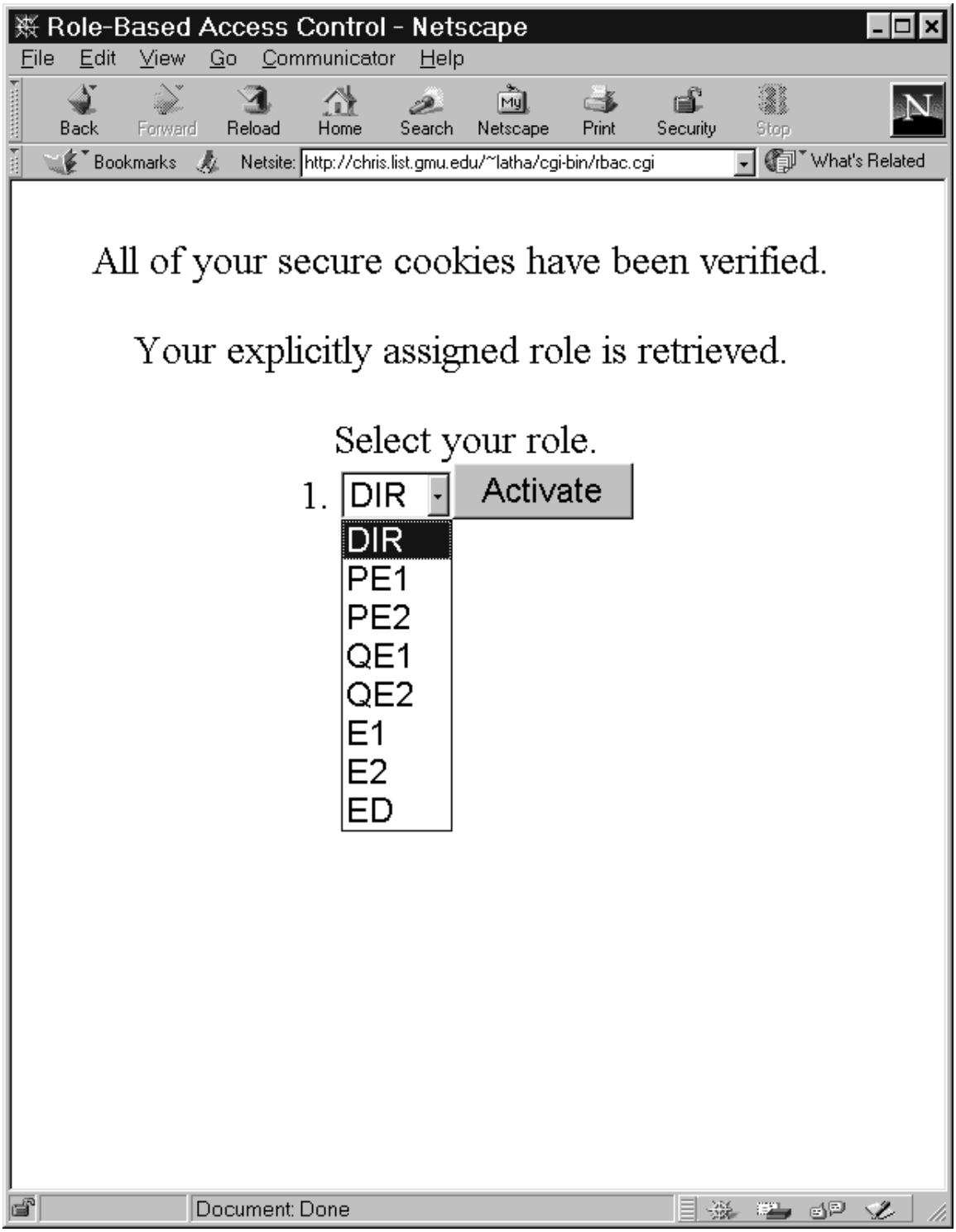


Figure A.10: An Example of Verification Success

APPENDIX B

SNAPSHOTS FROM RBAC ON THE WEB BY SMART CERTIFICATES

In this appendix, we attach the snapshots of selected images from our implementation of RBAC on the Web by smart certificates. The implementation is based on the role hierarchy depicted in Figure 6.8. In this example, Alice has four smart certificates in her browser, with each smart certificate containing different roles. Alice selects one of them according to her transactions in the beginning of the session, which requires a specific role. Detailed procedures are described in Section 6.3.

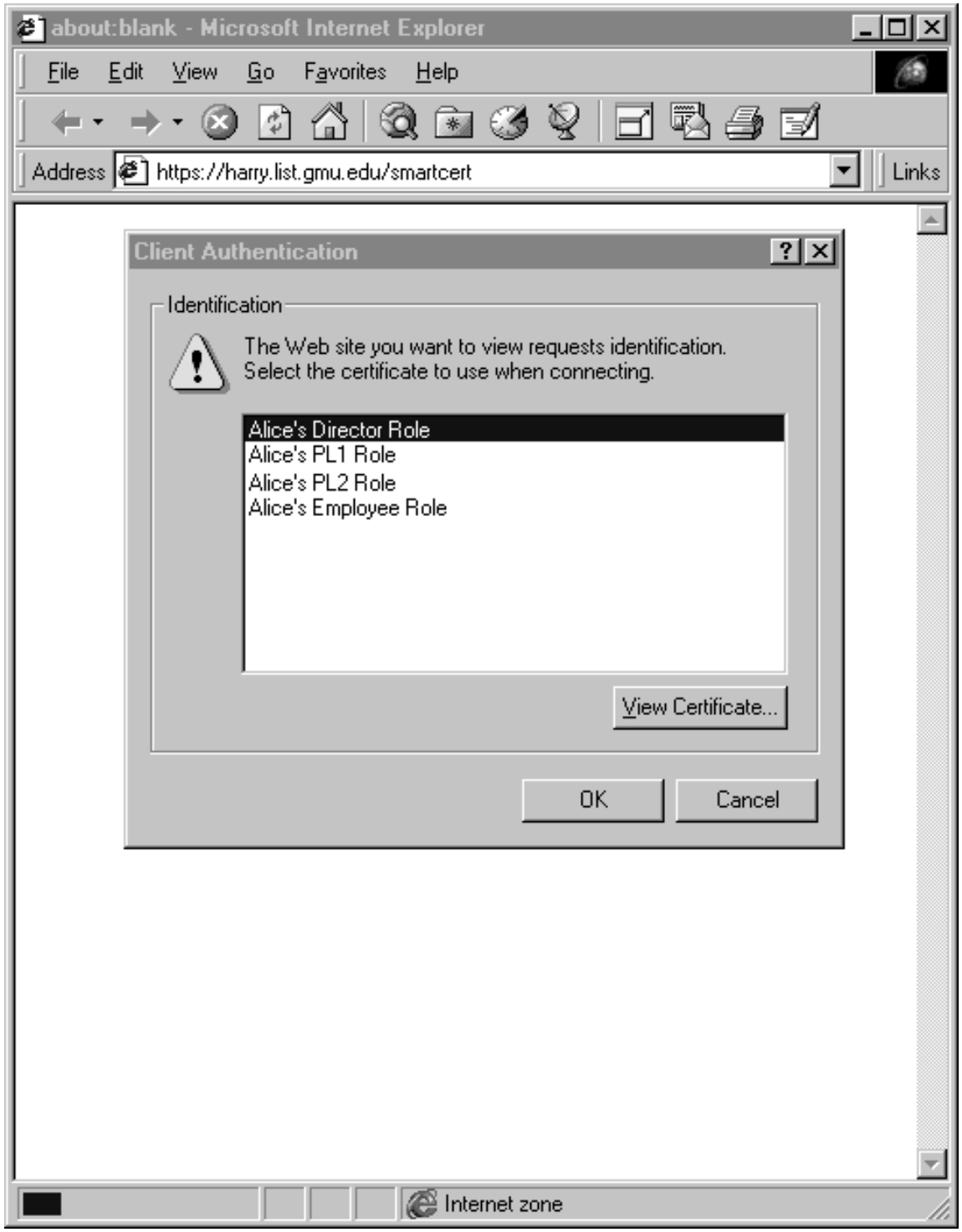


Figure B.1: Alice Selects the Smart Certificate for the Director Role

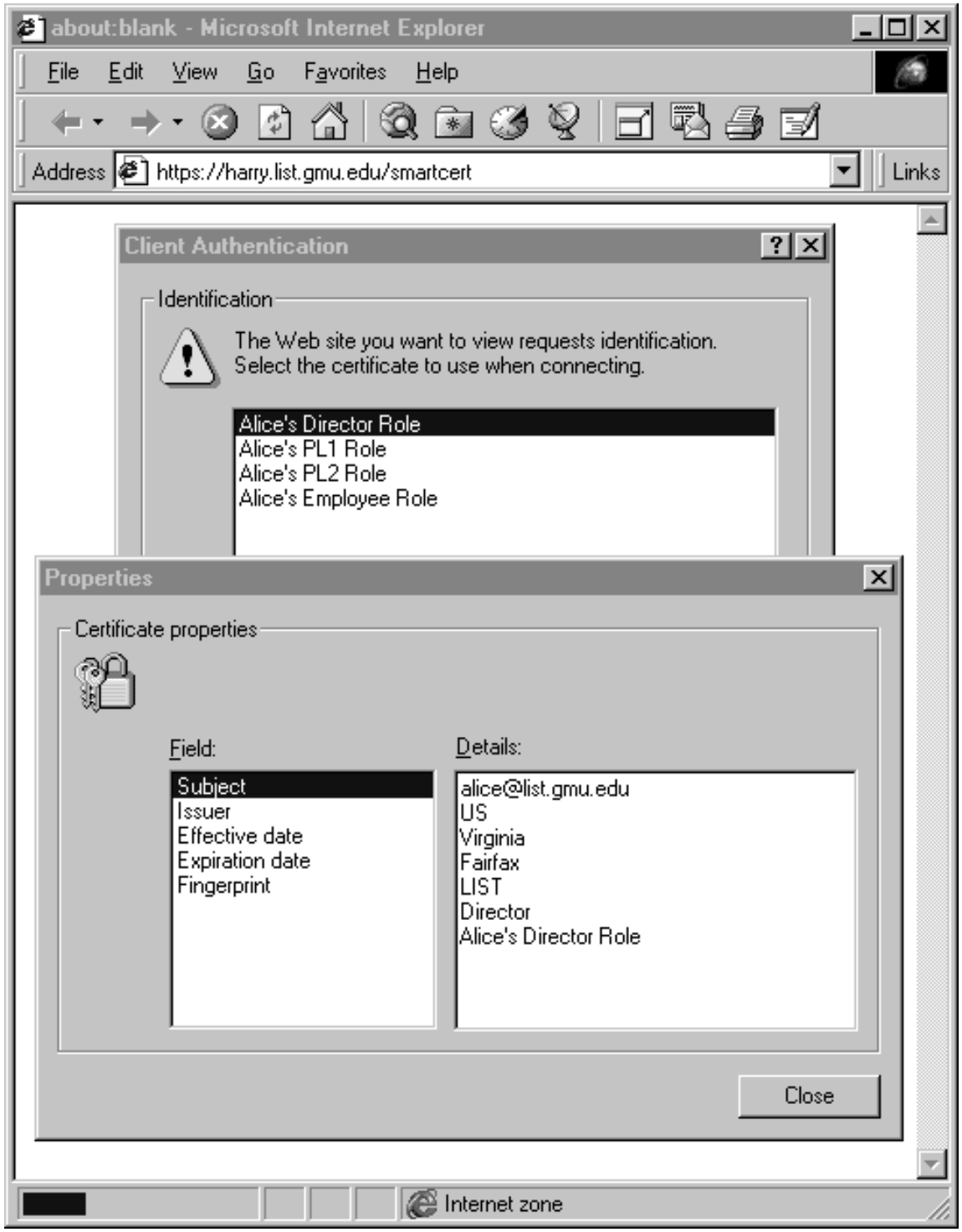


Figure B.2: The Contents of the Smart Certificate for the Director Role

Figure B.3: Site Introduction

Figure B.4: Alice Is Allowed to Access the Director's Page

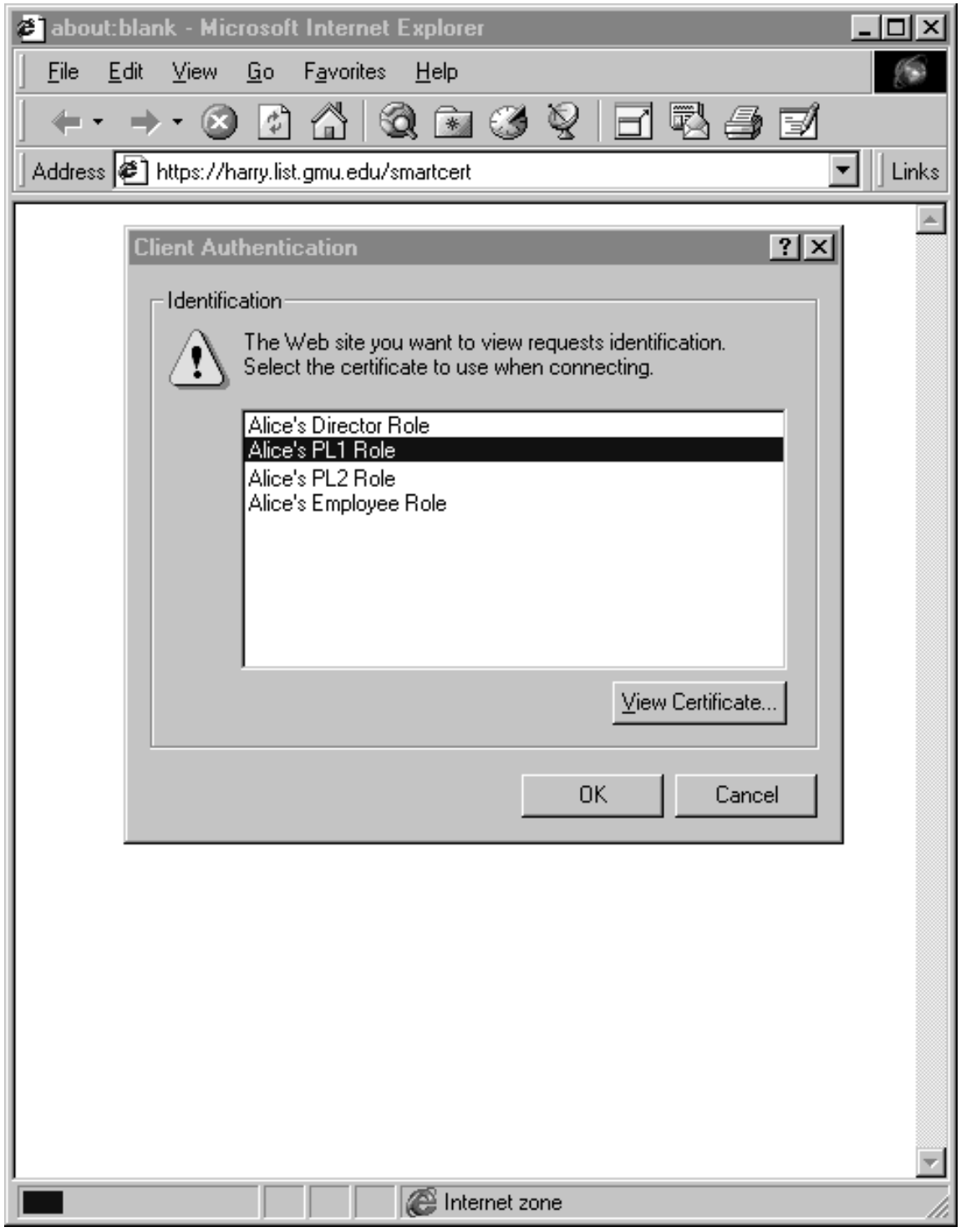


Figure B.5: Alice Selects the Smart Certificate for the PL1 Role

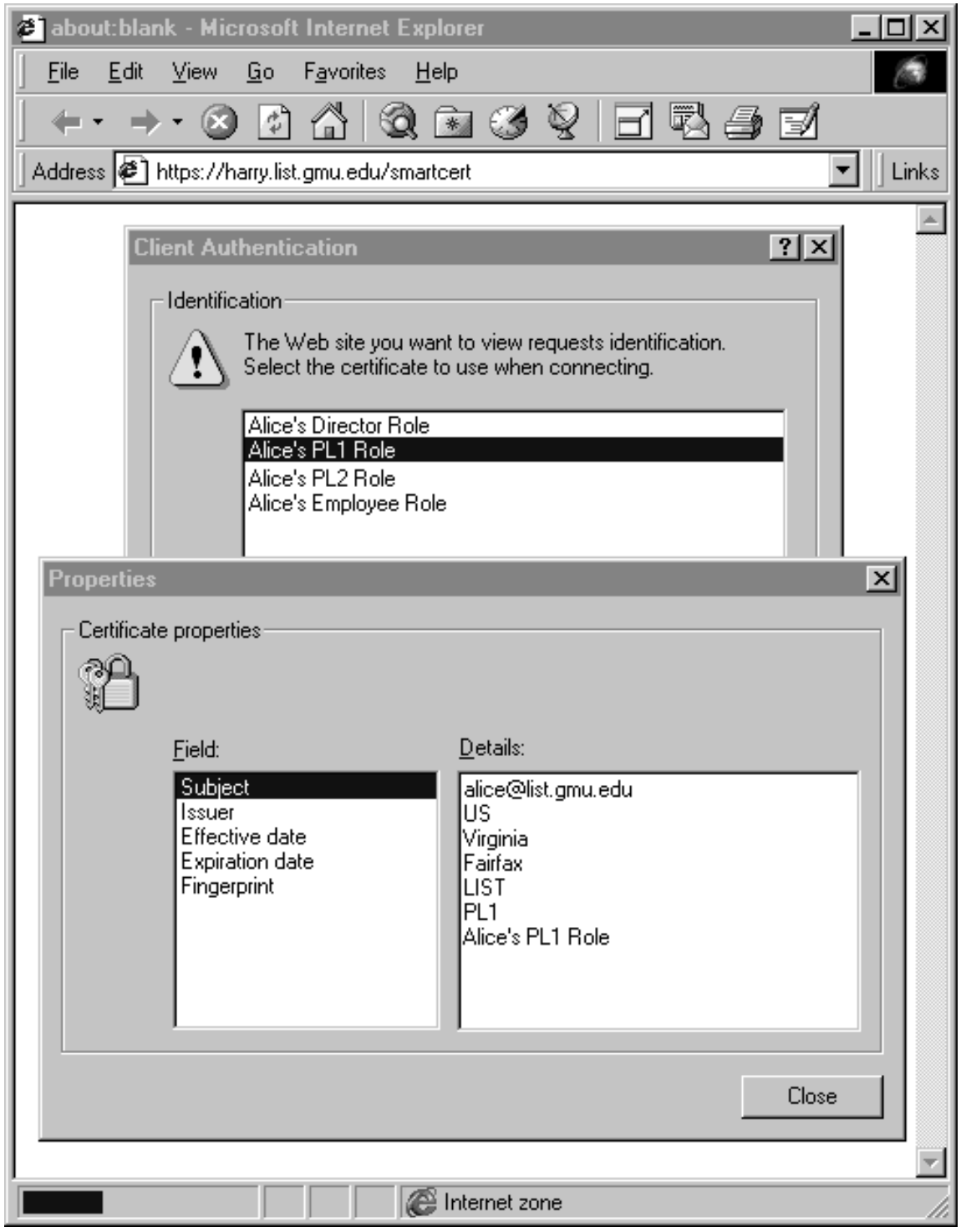


Figure B.6: The Contents of the Smart Certificate for the PL1 Role

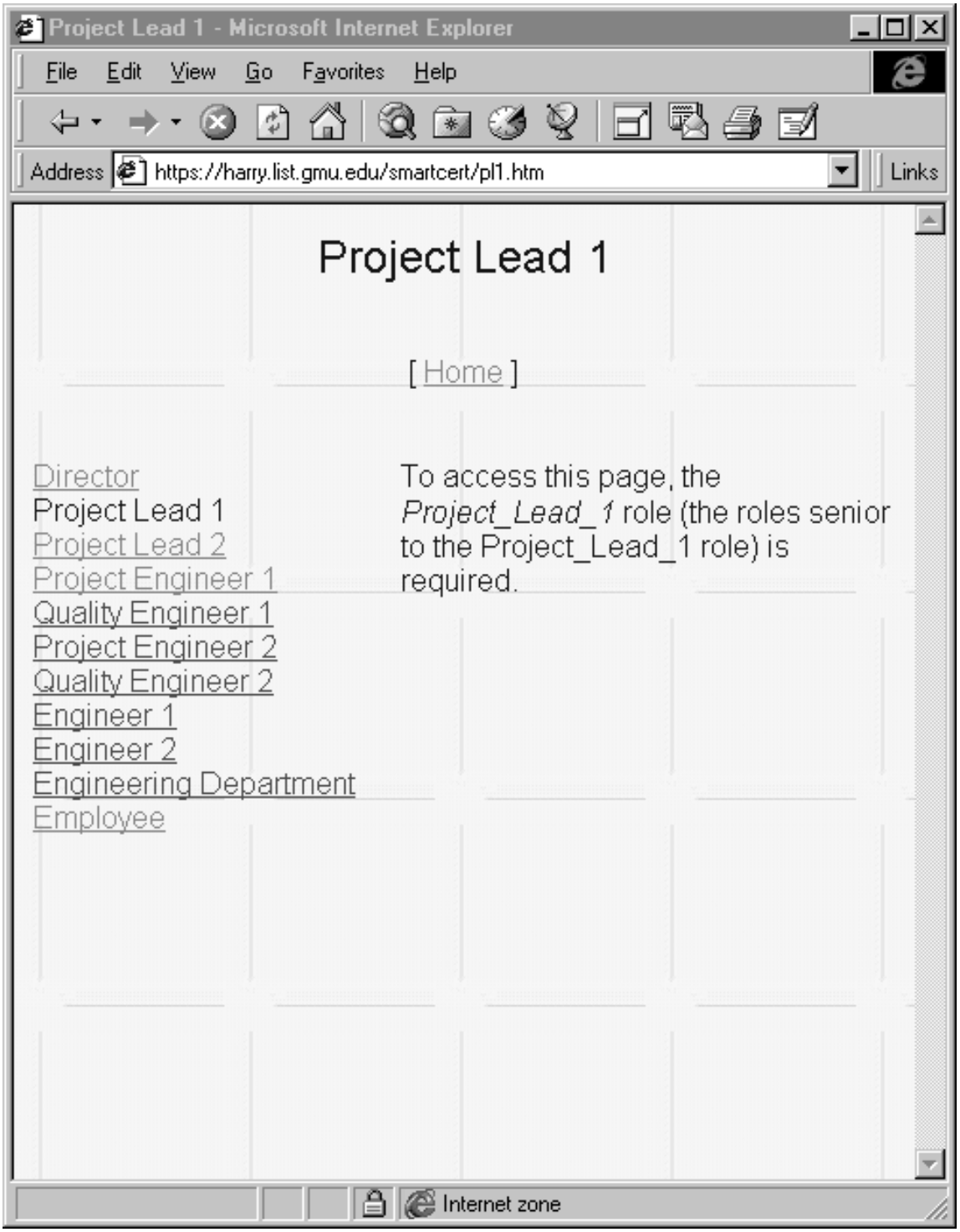


Figure B.7: Alice Is Allowed to Access the PL1's Page

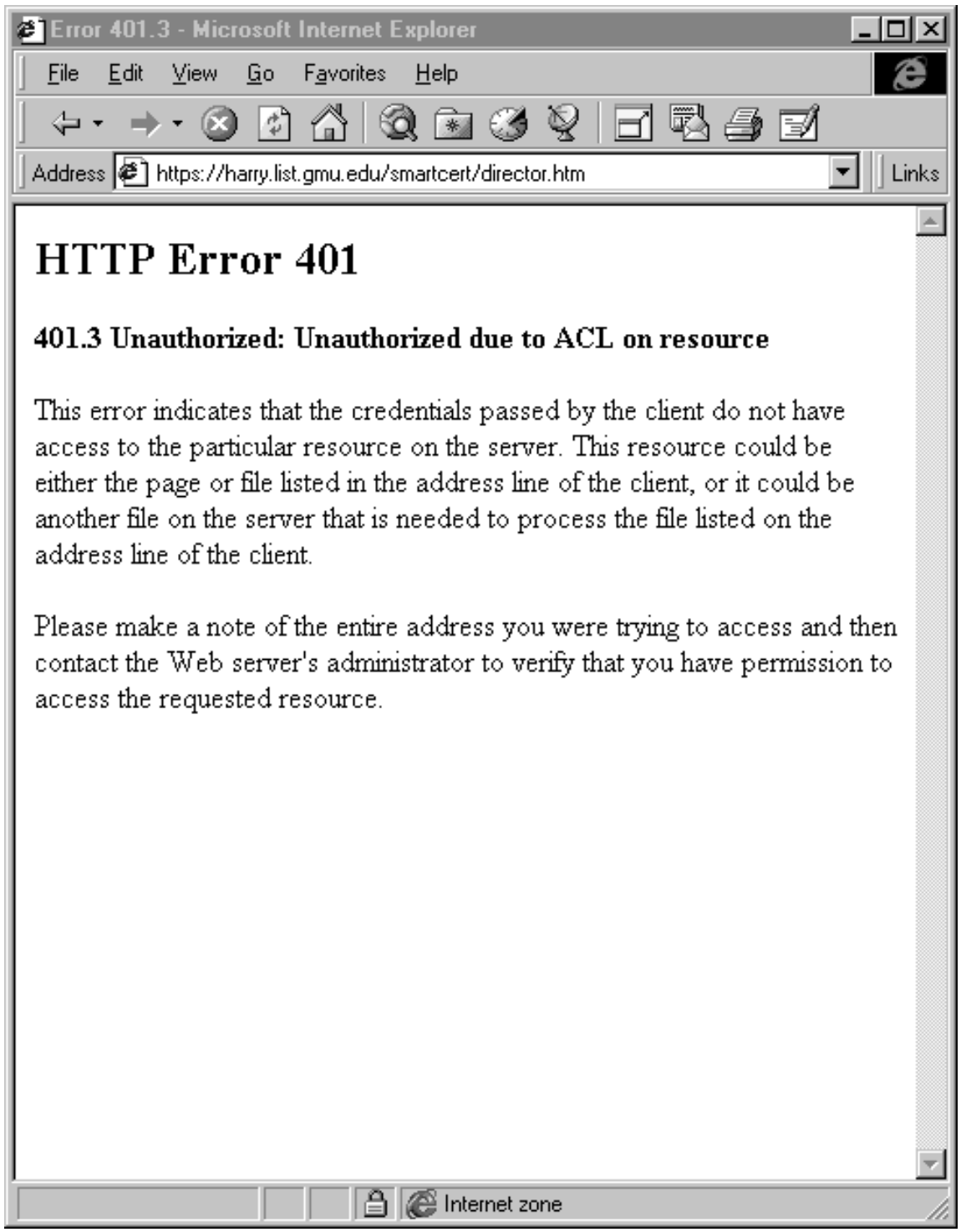


Figure B.8: Alice Is Not Allowed to Access the Director's Page

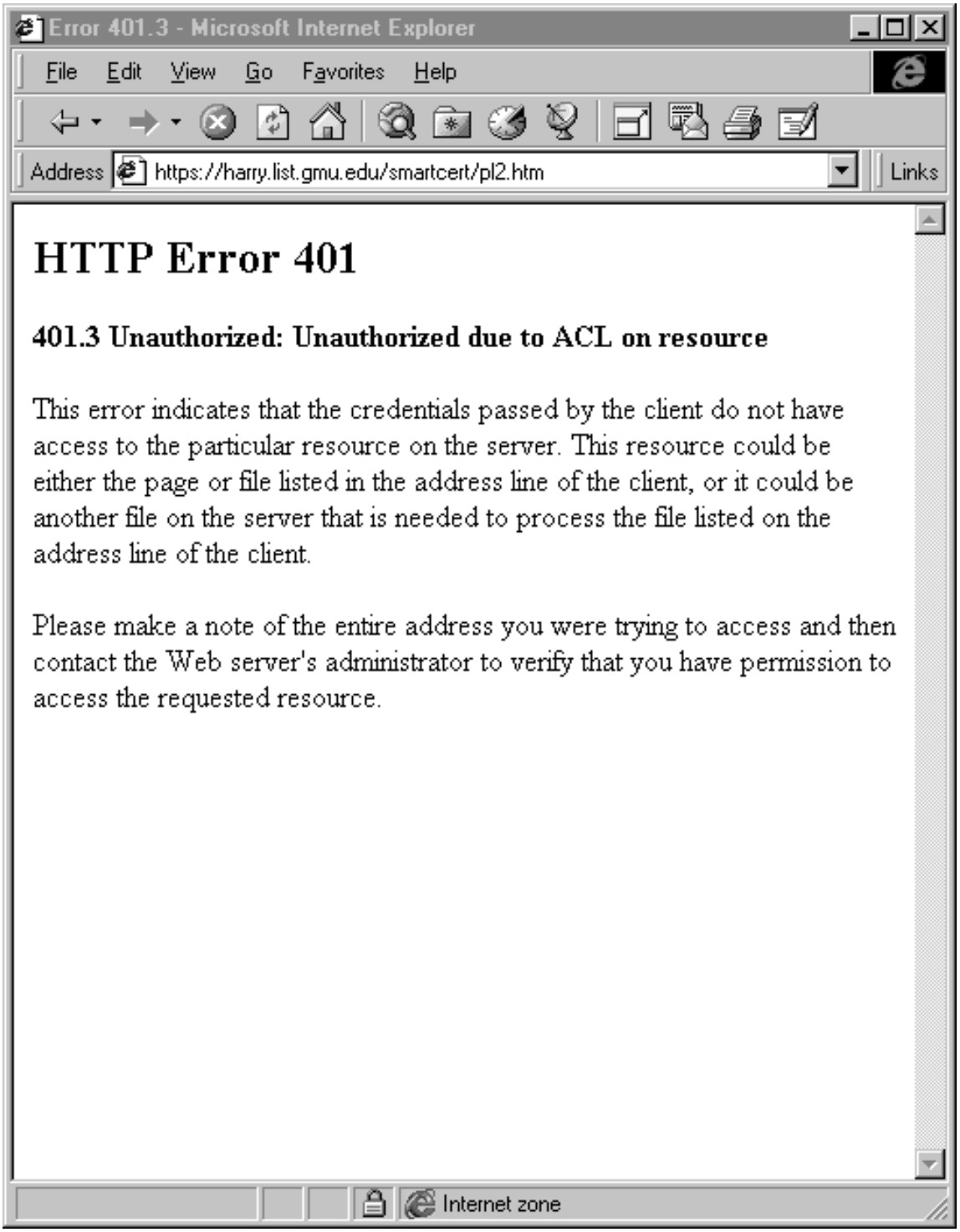


Figure B.9: Alice Is Not Allowed to Access the PL2's Page

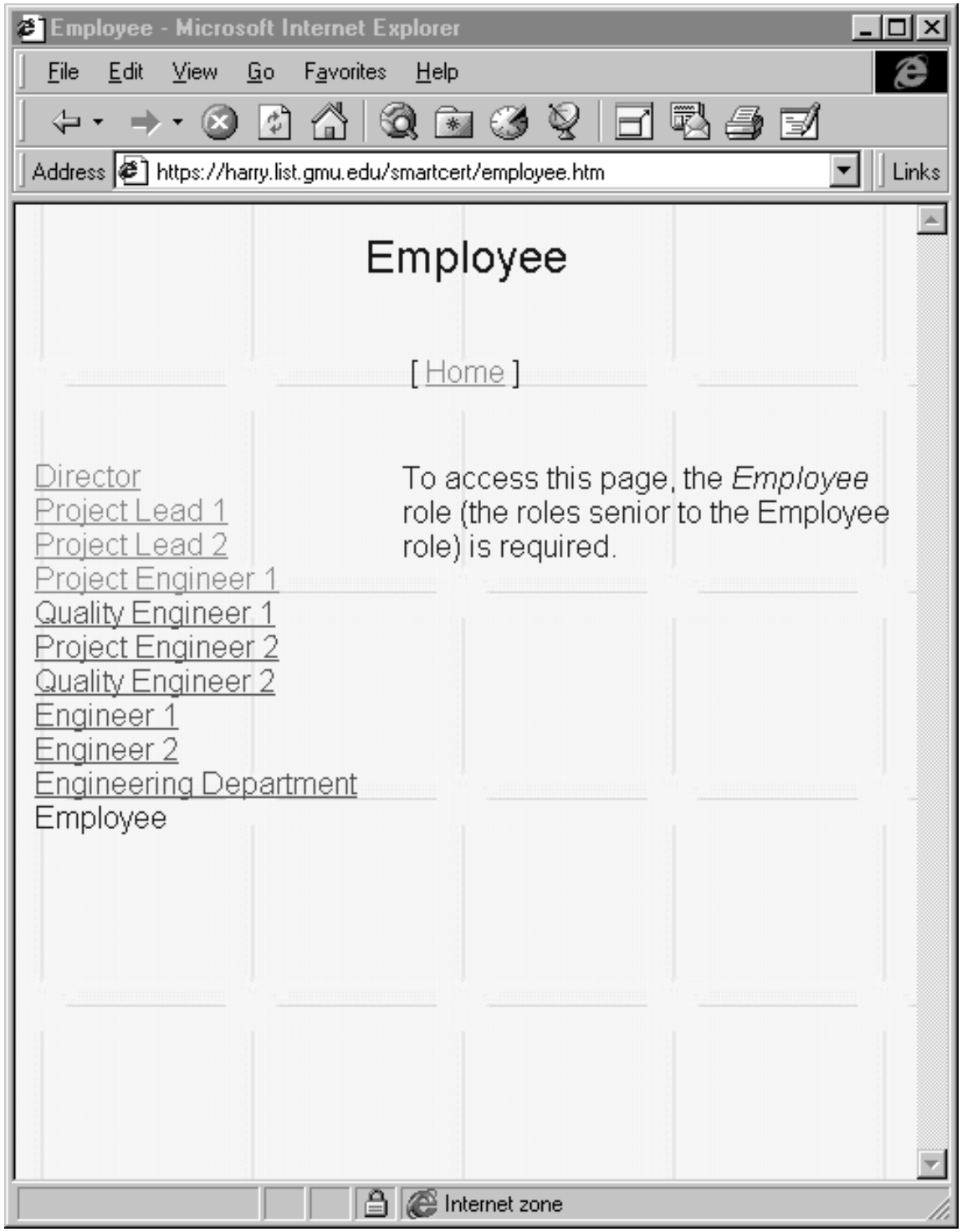


Figure B.10: Alice Is Allowed to Access the Employee's Page

BIBLIOGRAPHY

BIBLIOGRAPHY

[Bar97] Larry S. Bartz. hyperDRIVE: Leveraging LDAP to implement rbac on the web. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, pages 69–74. ACM, Fairfax, VA, November 6-7 1997.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hashing functions for message authentication. In *Advances in Cryptography - CRYPTO'90 Proceedings*, volume 1109, Springer-Verlag, 1996. Lecture Notes in Computer Science.

[BJR98] Grady Booch, Ivar Jacobson, and James Rumbaugh. *The unified modeling language user guide*. Addison-Wesley, 1998.

[CDFT98] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. *OpenPGP massage Format*, November 1998. RFC 2440.

[DA99] T. Dierks and C. Allen. *The TLS (Transport Layer Security) Protocol*, January 1999. RFC 246.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[DH97] Whitfield Diffie and Martin Hellman. *ANSI X9.42: Establishment of Symmetric Algorithm Keys Using Diffie-Hellman*, 1997. American National Standards Institute.

[enC98] enCommerce. *getAccess*, 1998. http://www.encommerce.com/products.

[Far98a] Stephen Farrell. *An Internet AttributeCertificate profile for Authorization*, August 1998. draft-ietf-tls-ac509prof-00.txt.

[Far98b] Stephen Farrell. *TLS extensions for AttributeCertificate based authorization*, February 1998. draft-ietf-tls-attr-cert-00.txt.

[FCK95] David Ferraiolo, Janet Cugini, and Richard Kuhn. Role-based access control (RBAC): Features and motivations. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 241–48, New Orleans, LA, December 11-15 1995.

[Fed91] Federal Register. *Proposed Federal Information Processing Standard for Digital Signature Standard (DSS)*, August 1991. n.169.

[Fed94] Federal Information Processing Standards Publication. *Digital Signature Standard (DSS)*, 1994. FIPS PUB 186.

[Fed95] Federal Information Processing Standard (FIPS). *Secure Hash Standard*, 1995. FIPS 180-1.

[FGM98] R. Fielding, J. Gettys, and J. C. Mogul. *HyperText Transfer Protocol (HTTP/1.1)*, November 1998. draft-ietf-http-v11-spec-rev-o6.

[FK92] David Ferraiolo and Richard Kuhn. Role-based access controls. In *Proceedings of 15th NIST-NCSC National Computer Security Conference*, pages 554–563, Baltimore, MD, October 13-16 1992.

[Gar95] Simson Garfinkel. *Pretty Good Privacy*. O'Reilly & Associates, 1995.

[GI96] Luigi Guiri and Pietro Iglio. A formal model for role-based access control with constraints. In *Proceedings of IEEE Computer Security Foundations Workshop 9*, pages 136–145, Kenmare, Ireland, June 1996.

[Gui95] Luigi Guiri. A new model for role-based access control. In *Proceedings of 11th Annual Computer Security Application Conference*, pages 249–255, New Orleans, LA, December 11-15 1995.

[HDT95] M.-Y. Hu, S.A. Demurjian, and T.C. Ting. User-role based security in the ADAM object-oriented design and analyses environment. In J. Biskup, M. Morgernstern, and C. Landwehr, editors, *Database Security VIII: Status and Prospects*. North-Holland, 1995.

[Her96] Eric Herrmann. *CGI Programming with Perl 5*. Sams Net, 1996.

[HFPS98] R. Housley, W. Ford, W. Polk, and D. Solo. *Internet X.509 public key infrastructure certificate and CRL profile*, September 1998. draft-ietf-pkix-ipki-part1-11.txt.

[HS98] Yung-Kao Hsu and Stephen P. Seymour. An internet security framework based on short-lived certificates. *IEEE Internet Computing*, pages 73–79, March/April 1998.

[ITU93] ITU-T Recommendation X.509. *Information technology - Open systems Interconnection - The Directory: Authentication Framework*, 1993. ISO/IEC 9594-8:1993.

[ITU97] ITU-T Recommendation X.509. *Information technology - Open systems Interconnection - The Directory: Authentication Framework*, 1997.

[KA98] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol*, November 1998. RFC 2401.

[KM97] David M. Kristol and Lou Montulli. *HTTP state management mechanism*, February 1997. RFC 2109.

[KM98a] David M. Kristol and Lou Montulli. *HTTP state management mechanism*, February 1998. draft-ietf-http-state-man-mec-8.txt.

[KM98b] David M. Kristol and Lou Montulli. *HTTP state management mechanism*, July 1998. draft-ietf-http-state-man-mec-10.txt.

[Lau98] Simon Laurent. *Cookies*. McGraw-Hill, 1998.

[LM91] X. Lai and J. Massey. A proposal for a new block encryption standard. In *Advances in Cryptography - CRYPTO'90 Proceedings*, pages 389–404, Springer-Verlag, 1991.

[MD94] Imtiaz Mohammed and David M. Dilts. Design for dynamic user-role-based security. *Computers & Security*, 13(8):661–671, 1994.

[Neu94] B. Clifford Neuman. Using Kerberos for authentication on computer networks. *IEEE Communications*, 32(9), 1994.

[Nix98] Siemens Nixdorf. *TrustedWeb*, 1998. http://www.sse.ie/TrustedWeb.

[NO95] Matunda Nyanchama and Sylvia Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgernstern, and C. Landwehr, editors, *Database Security VIII: Status and Prospects*. North-Holland, 1995.

[PP95] Tom Parker and Denis Pinkas. *SESAME V4 - OVERVIEW*. SESAME Technology, December 1995. Version 4.

[PS99a] Joon S. Park and Ravi Sandhu. RBAC on the web by smart certificates. In *Proceedings of 4th ACM Workshop on Role-Based Access Control*. ACM, Fairfax, VA, October 28-29 1999.

[PS99b] Joon S. Park and Ravi Sandhu. Smart certificates: Extending x.509 for secure attribute services on the Web. In *Proceedings of 22nd National Information Systems Security Conference*, Crystal City, VA, October 1999.

[PSG99] Joon S. Park, Ravi Sandhu, and SreeLatha Ghanta. RBAC on the Web by secure cookies. In *Proceedings of the IFIP WG11.3 Workshop on Database Security*. Chapman & Hall, July, 1999.

[Qua98] Terry Quatrani. *Visual Modeling with Rational Rose and UML*. Addison-Wesley, 1998.

[Riv92] R.L. Rivest. *The MD5 message digest algorithm*, April 1992. RFC 1321.

[RRSW97] C. Rigney, A. Rubens, W. A. Simpson, and S. Willens. *Remote Authentication Dial In User Service RADIUS*, April 1997. RFC 2138.

[RS98] E. Rescorla and A. Schiffman. *Security Extensions For HTML*, June 1998. draft-ietf-wts-shtml-05.txt.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[San97] Ravi Sandhu. Rationale for the RBAC96 family of access control models. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997.

[San98] Ravi Sandhu. Role-based access control. *Advances in Computers*, 46, 1998.

[SB97] Ravi Sandhu and Venkata Bhamidipati. The URA97 model for role-based administration of user-role assignment. In T. Y. Lin and Xiaolei Qian, editors, *Database Security XI: Status and Prospects*. North-Holland, 1997.

[SBC+97] Ravi Sandhu, Venkata Bhamidipati, Edward Coyne, Srinivas Ganta, and Charles Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and outline. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, pages 41–50. ACM, Fairfax, VA, November 6-7 1997.

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[SNS88] J.F. Steiner, C. Neuman, and J.I. Schiller. Kerberos: An authentication service for open network systems. In *Proc. Winter USENIX Conference*, 1988.

[SP98] Ravi Sandhu and Joon S. Park. Decentralized user-role assignment for Web-based intranets. In *Proceedings of 3rd ACM Workshop on Role-Based Access Control*, pages 1–12. ACM, Fairfax, VA, October 22-23 1998.

[SR98] A. Schiffman and E. Rescorla. *The Secure HyperText Transfer Protocol*, June 1998. draft-ietf-wts-shttp-06.txt.

[TC97] Zahir Tari and Shun-Wu Chan. A role-based access control for intranet security. *IEEE Internet Computing*, pages 24–34, September/October 1997.

[vSvdM94] S. H. von Solms and Isak van der Merwe. The management of computer security profiles using a role-oriented approach. *Computers & Security*, 13(8):673–680, 1994.

[WS96] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings of the Second UNIX Workshop on Electronic Commerce*, November 1996.

[YCS97] Charles Youman, Ed Coyne, and Ravi Sandhu, editors. *Proceedings of the 1st ACM Workshop on Role-Based Access Control, Nov 31-Dec. 1, 1995.* ACM, 1997.

[YHK95] W. Yeong, T. Howes, and S. Kille. *Lightweight Directory Access Protocol*, March 1995. RFC 1777.

[Zim95] Phillip R. Zimmermann. *The Official PGP User's Guide.* MIT Press, 1995.

CURRICULUM VITAE

Joon S. Park was born in Korea, in 1966. In 1989, he received a Bachelor of Science in Engineering from Yonsei University in Seoul, and in 1997, he obtained a Master of Science in Systems Engineering from George Mason University (GMU), in Fairfax, Virginia. He received a Ph.D in information technology, specializing in information security, from GMU in 1999. He worked for Goldstar (now LG) R&D Center as a research engineer, in Seoul, Korea, from 1989 to 1994. While completing his master's and doctoral study at GMU, he served in several research capacities for the university, beginning with work as a research assistant for GMU's Center for Software Systems Engineering (CSSE) in 1996. At CSSE, he provided technical research support to varied Navy and FBI projects, and he was later awarded the university's prestigious Technology Learning Competition prize. In 1996, he joined GMU's Laboratory for Information Security Technology (LIST). While pursuing his Ph.D. in Information Technology at GMU, he obtained a Certificate in Information Security in 1998. His doctoral work, primary addressing issues related to security services on the Web, has been generously sponsored by the National Institute of Standards and Technology (NIST), the National Science Foundation (NSF), the National Security Agency (NSA), and the Naval Research Laboratory (NRL).

This dissertation was typeset with LaTeX[‡] by the author.

[‡]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.