View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Royal Holloway Research Online

Interdomain User Authentication and Privacy

Andreas Pashalidis

Technical Report
RHUL–MA–2005–13
23 December 2005

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Abstract

This thesis looks at the issue of interdomain user authentication, i.e. user authentication in systems that extend over more than one administrative domain. It is divided into three parts. After a brief overview of related literature, the first part provides a taxonomy of current approaches to the problem. The taxonomy is first used to identify the relative strengths and weaknesses of each approach, and then employed as the basis for putting into context four concrete and novel schemes that are subsequently proposed in this part of the thesis. Three of these schemes build on existing technology; the first on 2nd and 3rd-generation cellular (mobile) telephony, the second on credit/debit smartcards, and the third on Trusted Computing. The fourth scheme is, in certain ways, different from the others. Most notably, unlike the other three schemes, it does not require the user to possess tamper-resistant hardware, and it is suitable for use from an untrusted access device. An implementation of the latter scheme (which works as a web proxy) is also described in this part of the thesis.

As the need to preserve one's privacy continues to gain importance in the digital world, it is important to enhance user authentication schemes with properties that enable users to remain anonymous (yet authenticated). In the second part of the thesis, anonymous credential systems are identified as a tool that can be used to achieve this goal. A formal model that captures relevant security and privacy notions for such systems is proposed. From this model, it is evident that there exist certain inherent limits to the privacy that such systems can offer. These are examined in more detail, and a scheme is proposed that mitigates the exposure to certain attacks that exploit these limits in order to compromise user privacy. The second part of the thesis also shows how to use an anonymous credential system in order to facilitate what we call 'privacy-aware single sign-on' in an open environment. The scheme enables the user to authenticate himself to service providers under separate identifier, where these identifiers cannot be linked to each other, even if all service providers collude. It is demonstrated that the anonymity enhancement scheme proposed earlier is particularly suited in this special application of anonymous credential systems.

Finally, the third part of the thesis concludes with some open research questions.

Interdomain User Authentication and Privacy

Andreas Pashalidis

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Information Security Group
Department of Mathematics
Royal Holloway, University of London
2005

Declaration

These doctoral studies were conducted under the supervision of Chris J. Mitchell and Peter Wild.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Andreas Pashalidis
June, 2005

Acknowledgements

I would like to thank my academic supervisor, Chris J. Mitchell, for his invaluable guidance and support throughout my studies at Royal Holloway. Without his insightful ideas, his top-quality comments and feedback on my work, our stimulating discussions and his constructive criticisms, this thesis would never come into existence. I am deeply grateful for the patience and dedication he showed to me. I am also indebted to my academic advisor, Peter Wild, for his continuous support throughout my studies.

I am grateful to all the lecturers, administrative and technical staff, and students in the Mathematics department. They provided me with an excellent research environment during my studies. Many people, some outside the department, have contributed to my studies. While everyone's contribution has been different, I feel that they have all been important. The list of people to whom I owe thanks is quite long (and is still changing). As any effort to write it down would be incomplete, I hope that they accept my apologies for not attempting to do so. You know who you are; thanks.

Finally, I owe thanks to the State Scholarship Foundation of Greece for sponsoring me during a significant part of my studies, and, most importantly, my family, my wife, and God for self-evident reasons.

Abstract

This thesis looks at the issue of interdomain user authentication, i.e. user authentication in systems that extend over more than one administrative domain. It is divided into three parts. After a brief overview of related literature, the first part provides a taxonomy of current approaches to the problem. The taxonomy is first used to identify the relative strengths and weaknesses of each approach, and then employed as the basis for putting into context four concrete and novel schemes that are subsequently proposed in this part of the thesis. Three of these schemes build on existing technology; the first on 2nd and 3rd-generation cellular (mobile) telephony, the second on credit/debit smartcards, and the third on Trusted Computing. The fourth scheme is, in certain ways, different from the others. Most notably, unlike the other three schemes, it does not require the user to possess tamper-resistant hardware, and it is suitable for use from an *untrusted* access device. An implementation of the latter scheme (which works as a web proxy) is also described in this part of the thesis.

As the need to preserve one's privacy continues to gain importance in the digital world, it is important to enhance user authentication schemes with properties that enable users to remain anonymous (yet authenticated). In the second part of the thesis, anonymous credential systems are identified as a tool that can be used to achieve this goal. A formal model that captures relevant security and privacy notions for such systems is proposed. From this model, it is evident that there exist certain inherent limits to the privacy that such systems can offer. These are examined in more detail, and a scheme is proposed that mitigates the exposure to certain attacks that exploit these limits in order to compromise user privacy. The second part of the thesis also shows how to use an anonymous credential system in order to facilitate what we call 'privacy-aware single sign-on' in an open environment. The scheme enables the user to authenticate himself to service providers under separate identifier, where these identifiers cannot be linked to each other, even if all service providers collude. It is demonstrated that the anonymity enhancement scheme proposed earlier is particularly suited in this special application of anonymous credential systems.

Finally, the third part of the thesis concludes with some open research questions.

Contents

List of Figures

List of Tables

Abbreviations

AA:	Authentication Application
AS:	Authentication Service
AuC:	Authentication Centre
BIOS:	Basic Input Output System
CA:	Certification Authority
CS:	Cardholder System
DAA:	Direct Anonymous Attestation
DDA:	Dynamic Data Authentication
DNS:	Domain Name Service
EK:	Endorsement Key
EMV:	Europay Mastercard Visa
FTP:	File Transfer Protocol
GSM:	Global System for Mobile Communications
HTTP:	Hypertext Transfer Protocol
HTTPS:	Hypertext Transfer Protocol Security
ICC:	Integrated Circuit Card
IMSI:	International Mobile Subscriber Identity
MAC:	Message Authentication Code
ME:	Mobile Equipment
PAPTS:	Privacy-Aware Proxy-based True Single sign-on
PCR:	Platform Configuration Register
PIN:	Personal Identification Number
PKI:	Public Key Infrastructure
PRV-CA:	Privacy Certification Authority
PVDE:	PIN Verification Data Element
SAML:	Security Assertion Markup Language
SIM:	Subscriber Identity Module
SP:	Service Provider
SPID:	Service Provider Identifier
SSL:	Secure Sockets Layer
SSO:	Single Sign-On
TCG:	Trusted Computing Group
TLS:	Transport Layer Security
TP:	Trusted Platform
TPM:	Trusted Platform Module
TTP:	Trusted Third Party
UMTS:	Universal Mobile Telecommunications System
URL:	Uniform Resource Locator
USB:	Universal Serial Bus

Chapter 1

Introduction

Contents

This chapter briefly introduces the concept of interdomain user authentication and introduces a number of security constructs which the remainder of the thesis makes use of.

In many situations, users are required to identify themselves in order to access some service or resource. The identification takes place by means of a piece of information, called an *identifier*, that is unique to each user. Sometimes it is not sufficient for the user to simply provide an identifier and claim to be its legitimate owner; the user is required to somehow convince the service provider that he is indeed the legitimate owner of the stated identifier, as claimed. This process, called *authentication*, is achieved by means of an authentication *scheme*. Most such schemes require the user to maintain and have available some additional information. This information, called the user's *authentication credential*, is typically designed to be user-specific and hard for others to guess or replicate[1.1]. As such, it allows the user to authenticate himself as the legitimate owner of the associated identifer. The user typically needs to maintain an authentication credential for each of his identifiers.

Different service providers may have different ways to identify users, and use different methods to authenticate the corresponding identifiers. As the number of service providers a user has a relationship with grows, so does the number of his identifiers and thereby the burden imposed on him by the requirement to maintain the associated authentication credentials.

[1.1] Perhaps the most common user authentication credential is a username/password pair.

The purpose of the schemes considered in this thesis is to relieve the user from this burden. There are three distinct advantages one can potentially gain by using such a scheme. Firstly, one can gain a usability advantage, because the scheme removes the necessity for the user to (manually) maintain as many authentication credentials as the number of service providers with which he has a relationship. Secondly, the management of user accounts can be simplified in certain scenarios. Thirdly, the overall level of security is potentially increased because the user is required to maintain a smaller number of authentication credentials and he may be able do so securely with a reasonable amount of effort.

A scheme that enables the user to authenticate himself to more than one service provider using only a single authentication credential, is also called a 'Single Sign-On' (SSO) scheme. We use this term throughout the thesis.

1.1 Security constructs and terminology used throughout the thesis

This section gives a brief and informal introduction to certain well-established cryptographic terminology and mechanisms used throughout the thesis. We do not analyse these constructs and their operation here; the reader interested in a more comprehensive introduction is referred to [142, 188].

1.1.1 Nonces

The term 'nonce' means 'number used once'. It is a finite sequence of bits that is used only once within a given context, and is discarded afterwards. Nonces are typically employed by entity authentication schemes in order to provide a guarantee of freshness, or 'liveness', to an interested party. This works as follows. The interested party A generates a nonce and sends it to the party (B) of whose liveness A wants to be convinced. Party B then performs a computation with the nonce and returns the result to A[1.2]. Party A checks the correctness of the result; if it is correct, A concludes that his communication partner is 'alive', i.e. must have responded to A's original message.

[1.2]This computation is typically of a cryptographic nature and may involve other data, such as keys.

It is a typical requirement that the nonces employed in the execution of an authentication protocol should never occur again in future executions. Furthermore, in order to prevent certain types of attack, nonces typically have to be hard to guess by an attacker. Typically, nonces are randomly generated when needed. Some of the protocols introduced in this thesis make use of nonces.

1.1.2 One-way hash functions

A one-way hash function h is a function that maps an arbitrary-length message m to a fixed-size sequence of bits, called a *digest* $d = h(m)$. A secure one way hash function is required to satisfy the following requirements.

- (Being one way:) Given d it should be infeasible to find an m such that $d = h(m)$.
- (Second pre-image resistance:) Given any m it should be infeasible to find an $m' \neq m$ such that $h(m') = h(m)$.

An additional property that one way functions may be required to satisfy is that of 'collision resistance'. This property states that it should be infeasible to find values m, $m'(m \neq m')$ such that $h(m) = h(m')$.

More rigorous definitions, as well as examples of one way hash functions, can be found in [139, 142, 188]. A widely used hash function, called the 'Secure Hash Algorithm 1' (SHA-1) is specified in [147]. It is worth noting that one way hash functions are a fundamental building block for many cryptosystems. In order to facilitate the analysis of such systems in the setting of 'provable security', a theoretic abstraction of one way hash functions, called the 'random oracle model', has been proposed and used in the literature. The interested reader is referred to [16, 42]. Whenever we use a one way hash function in this thesis, we assume that it is collision-resistant.

1.1.3 Key derivation functions

A key derivation function is a function that generates a cryptographic key from input 'keying material'. This keying material typically consists of a secret value such as another crypto-

graphic key, a password, or the result of some cryptographic computation, and may also include material such as an algorithm identifier or a random value. Key derivation functions are typically based on one way hash functions or on pseudo-random functions. A specification for password-based key derivation can be found in [100]. The reader interested in pseudo-random functions is referred to [11].

The remaining security constructs we use can be classified as either symmetric or asymmetric cryptosystems.

1.1.4 Symmetric cryptosystems

Symmetric cryptosystems, in general, make use of a single finite sequence of bits, called a 'key'. This key is typically shared among, and kept secret by, two or more communicating partners.

1.1.4.1 Encryption schemes

A symmetric encryption scheme provides two algorithms, namely `encrypt` and `decrypt`. The `encrypt` algorithm takes as input a plaintext message m and a key k and outputs a ciphertext message c. The `decrypt` algorithm takes as input a ciphertext message $\bar{c}$ and a key $\bar{k}$ and outputs a plaintext message $\bar{m}$. For an encryption scheme it should hold that

- if $\bar{c} = c$ and $\bar{k} = k$, the `decrypt` algorithm outputs $\bar{m} = m$, and
- without knowledge of k, and given c, it is infeasible to infer any information about the message m (apart from its length).

In practice, encryption schemes are used to protect the *confidentiality* of a message. There exist several, well defined, security notions for symmetric encryption schemes [13, 111].

1.1.4.2 Message authentication codes

A message authentication code (MAC) scheme provides two algorithms, namely `compute` and `verify`. The `compute` algorithm, on input a message m and a key k, outputs a MAC μ. The `verify` algorithm, on input a message $\bar{m}$, a key $\bar{k}$ and a MAC $\bar{\mu}$, outputs either `accept` or `reject`. For a secure MAC scheme it should hold that

- if $\bar{m} = m$, $\bar{k} = k$ and $\bar{\mu} = \mu$ the `verify` algorithm outputs `accept`, and
- without knowledge of $\bar{k}$, it is infeasible to generate a message/MAC pair $(\bar{m}, \bar{\mu})$ such that the `verify` algorithm outputs `accept`.

A stronger notion of security for a MAC scheme, which is called 'existential unforgeability', is said to be satisfied if the second of the above conditions holds even if one is given access to an 'oracle' that outputs μ for messages m of one's choosing, and where $\bar{m}$ is different from all m that were given to the oracle. A MAC scheme that satisfies this notion is said to be 'existentially unforgeable'.

In practice, MACs are used in order to protect the *integrity* of a message. Several schemes exist and have been standardised [67]. The security of MAC schemes in general is discussed in more detail in [15, 142]. Whenever we make use of a MAC scheme in this thesis, we will indicate which security notion it is required to satisfy.

It is perhaps worth noting at this point that certain subtle issues arise when, in order to simultaneously protect both the integrity and the confidentiality for some message, one combines encryption and a MAC. The interested reader is referred to [15]. This thesis does not address such issues.

1.1.5 Asymmetric cryptosystems

Asymmetric cryptosystems typically require the participants to have a key *pair* consisting of a private and a public key. As its name suggests, the public key can be, and typically is, made available to other participants in the system. In this setting it is crucial for the participants to be able to check the authenticity of the public keys they receive. This is

because otherwise it might be possible for an adversary to falsify public keys without being detected.

One well-established mechanism that enables one to verify the authenticity of a received public key is a Public Key Infrastructure (PKI). In a PKI, public keys are embedded in 'certificates' that are issued by a special authority, called a 'Certification Authority' (CA). A certificate is a statement that binds an identifier to the public key; it is signed using a digital signature scheme (see below) and attests to the fact that the CA vouches for this binding. The recipient of a certificate can verify the CA's signature and is thereby assured that the public key belongs to the participant that is represented by the embedded name. A widely used standard for the format of certificates is described in [104]. The reader interested in PKIs is referred to [82].

1.1.5.1 Encryption Schemes

An asymmetric encryption scheme provides three algorithms, namely `generate`, `encrypt` and `decrypt`. The `generate` algorithm is used to produce a new key pair. Its outputs, denoted *pub* and *prv*, are a public and a private key respectively. These keys are related to each other; we denote this relationship as $prv \sim pub$. The `encrypt` algorithm takes as input a plaintext message m and a public key *pub* and outputs a ciphertext message c. The `decrypt` algorithm takes as input a ciphertext message $\bar{c}$ and a private key *prv* and outputs a plaintext message $\bar{m}$. For an asymmetric encryption scheme it should hold that

- if $\bar{c} = c$, the `decrypt` algorithm outputs $\bar{m} = m$ if $prv \sim pub$.
- without knowledge of *prv* and given c, it is infeasible to infer any information about m (apart from its length).

In other words, one should be able to correctly decrypt a message that was encrypted under a public key, only when one knows the *corresponding* private key.

In practice, asymmetric encryption schemes are used to protect the *confidentiality* of a message. One of the most important differences between symmetric and asymmetric encryption schemes is that, in an asymmetric scheme, the encrypter of a message does not need to

have access to the decryption key. There exist several, well defined, security notions for asymmetric encryption schemes [14].

1.1.5.2 Signature schemes

A signature scheme provides three algorithms, namely `generate`, `sign` and `verify`. The `generate` algorithm is used to produce a new key pair. Its outputs, denoted pub and prv, are a public and a private key respectively. These keys are related to each other; we denote this relationship as $pub \sim prv$. The `sign` algorithm takes as input a message m and a private key prv and outputs a finite sequence of bits, called a signature σ. The `verify` algorithm takes as input a message $\bar{m}$, a public key pub and a signature $\bar{\sigma}$ and outputs either `accept` or `reject`. For a secure signature scheme it should hold that

- if $\bar{m} = m$, $pub \sim prv$ and $\bar{\sigma} = \sigma$, then the `verify` algorithm outputs `accept`
- without knowledge of prv such that $pub \sim prv$, it is infeasible to generate a message/signature pair $(\bar{m}, \bar{\sigma})$ such that the `verify` algorithm outputs `accept`.

In other words, a signature should only verify successfully if it is evaluated on the same message as it was computed by the `sign` algorithm and the public key that corresponds to the private key input to the `sign` algorithm.

In practice, signature schemes are used to protect the *integrity* of a message. Additionally, they provide a form of *message authentication*, since a signature that verifies correctly under a given public key can only be computed by the holder of the corresponding private key. As long as the signer keeps this private key secret, the verifier of a signed message can be confident that the message was indeed signed by the signer. Furthermore, under certain circumstances, signature schemes can provide a *non-repudiation* service. This is because, if a signature that allegedly verifies correctly under some public key pub is disputed by a signer, a trusted arbitrator can be asked to independently verify the signature. If the verification is successful, the arbitrator is also required to establish by some means that the corresponding private key $prv \sim pub$ was indeed known only to the signer at the time the signature was generated. Demonstrating the above may then be treated as strong evidence that the signer indeed generated the signature and that, as such, he is liable for any commitments arising from the signed message.

There exist several well defined security notions for signature schemes [90].

1.1.5.3 Blind signatures

A blind signature scheme provides five algorithms, namely `generate`, `blind`, `sign`, `unblind`, and `verify`. The `generate` algorithm is used to produce a new key pair. Its outputs, denoted *pub* and *prv*, are a public and a private key respectively. These keys are related to each other; we denote this relationship as $pub \sim prv$. The `blind` algorithm takes as input a message m and a random number r, and outputs a *blinded* message m'. The `sign` algorithm takes as input a blinded message m' and a private key *prv* and outputs a finite sequence of bits, called a blind signature σ'. The `unblind` algorithm takes as input a blind signature σ', and an auxiliary input r' (and possibly a blinded message m'), and outputs an unblinded signature σ. The `verify` algorithm is similar to that of a conventional signature scheme, i.e. it takes as input a message $\bar{m}$, a public key *pub* and a signature $\bar{\sigma}$ and outputs either `accept` or `reject`. For a secure blind signature scheme it should hold that

- if $\bar{m} = m$, $r = r'$, $pub \sim prv$ and $\bar{\sigma} = \sigma$, then the `verify` algorithm outputs `accept`
- without knowledge of *prv* such that $pub \sim prv$, it is infeasible to generate a message/signature pair $(\bar{m}, \bar{\sigma})$ such that the `verify` algorithm outputs `accept`.
- given a blinded message/signature pair (m', σ'), without knowledge of r, it is infeasible to generate a message/signature pair $(\bar{m}, \bar{\sigma})$ such that the `verify` algorithm outputs `accept`.
- the blinded message/signature pair (m', σ') does not reveal any information about the unblinded message/signature pair (m, σ).

Blind signature schemes are typically used in settings where one party, say A, wishes to obtain a signature from another party, say B, in such a way that B does not get to know anything about the (unblinded version of the) message that is signed. This is typically achieved as follows. First, A generates the message m and blinds it using the `blind` algorithm. The resulting blinded message m' is sent to B which signs it using the `sign` algorithm. The resulting blind signature σ' is returned to A which now can convert it into a valid signature σ on the original message m using the `unblind` algorithm.

A property of blind signature schemes is that, if A later shows the message/signature pair (m, σ) to B (or a third party that may cooperate with B), then, assuming that in the meantime B has issued multiple blind signatures, B cannot link this pair with a particular previously signed blinded message. This protects A's privacy.

There exist several blind signature schemes and well defined security notions for such schemes (see, for example, [46, 110, 124, 169]). It is perhaps worth noting that other types of signature scheme have been proposed over the years. These schemes have different properties and include fail-stop signatures (see, e.g., [162]), forward-secure signatures (see, e.g., [121]), group signatures (see, e.g., [19]), proxy signatures (see, e.g., [123]), undeniable signatures (see, e.g., [84]) and others (see, e.g., [26]). Unless explicitly stated, this thesis makes use of conventional signature schemes (as described in section 1.1.5.2). Whenever we use such a scheme in this thesis, we indicate which security notion it is required to satisfy.

1.1.5.4 Zero-knowledge proof systems

A zero-knowledge proof system is a protocol between two parties, called the prover and the verifier, whereby the prover can convince the verifier about the validity of a statement, in such a way that the verifier gains no knowledge beyond the fact that the statement is valid (or invalid). Some zero-knowledge proof systems are interactive. In such systems the prover and the verifier exchange a number of messages throughout the execution of the protocol. Other zero-knowledge proof systems are non-interactive. In such systems, the proof is simply a string of bits that the prover sends to the verifier.

In this thesis, zero-knowledge proof systems are used in order for a prover to convince a verifier about the validity of a statement of the form "I know a valid signature on a message". A proof for such a statement is also known as a 'zero-knowledge proof of knowledge' of the signature. In order to verify the proof, the verifier needs to have access to the message and the public key that corresponds to the private key that was used to generate the signature. Statements of this form often occur in the context of anonymous credential systems, which are discussed in greater detail in part II of this thesis. It is important to note that, using a zero-knowledge proof system in conjunction with statements of the above form, the verifier does not get to know anything about the signature that is known by the prover. This is in contrast to a conventional signature scheme that requires the signature to be revealed to the party that verifies its validity.

For more rigorous definitions and security properties that apply to zero-knowledge proof systems the reader is referred to [87, 88].

1.2 Publications and origins of contributions

This thesis contains previous research that has been published in the proceedings of a number of refereed conferences, as follows.

- A. Pashalidis and C. J. Mitchell. *Impostor: A Single Sign-on System for Use from Untrusted Devices*. IEEE Globecom 2004 Conference, Dallas, Texas, USA, November 29 – December 3, 2004.
- A. Pashalidis and C.J. Mitchell, *A Security Model for Anonymous Credential Systems*, in Y. Deswarte, F. Cuppens, S. Jajodia and L. Wang (editors), Information Security Management, Education and Privacy, Proceedings of the 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems (I-NetSec'04), Kluwer Academic Publishers, pages 183–199, Toulouse, France, August 2004.
- A. Pashalidis and C.J. Mitchell, *Using EVM Cards for Single Sign-On*, in Sokratis K. Katsikas, Stefanos Gritzalis and Javier Lopez (editors), Proceedings of the First European PKI Workshop: Research and Applications, EuroPKI 2004, Samos Island, Greece, June 25-26 2004, Springer Verlag (LNCS 3093), Berlin, pp. 205–217.
- A. Pashalidis, *A Cautionary Note on Automatic Proxy Configuration*, in M.H. Hamza (editor), Proceedings of the IASTED International Conference on Communication, Network, and Information Security CNIS 2003, December 10-12, New York, USA, pp. 153–158.
- A. Pashalidis and C. J. Mitchell, *Using GSM/UMTS for Single Sign-On*, in Proceedings of SympoTIC '03, Joint IST Workshop on Mobile Future and Symposium on Trends in Communications, Bratislava, Slovakia, October 2003, IEEE Press, 2003, pp. 138–145.
- A. Pashalidis and C. J. Mitchell, *Single Sign-On Using Trusted Platforms*, in C. Boyd and W. Mao (editors), Proceedings of the 6th International Information Security Conference (ISC 2003), Bristol, UK, October 2003, Proceedings, Springer-Verlag (LNCS 2851), Berlin, pp. 54–68.

Other research that is included in this thesis has been published in a non-refereed fashion, as follows.

- A. Pashalidis and C. J. Mitchell, *Limits to Anonymity when Using Credentials*, to appear in Proceedings of the 12th International Workshop on Security Protocols, Cambridge, U.K., March 2004 Springer Verlag (LNCS), Berlin.
- A. Pashalidis and C. J. Mitchell, *A Taxonomy of Single Sign-On Systems*, in R. Safavi-Naini and J. Seberry (editors), Information Security and Privacy – 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11 2003, Proceedings, Springer-Verlag (LNCS 2727), Berlin (2003), pp. 249–264.
- A. Pashalidis and C. J. Mitchell, *Single Sign-On using Trusted Computing*, chapter 6 of TCG-conformant platforms, Chris J. Mitchell (editor), 2005, IEE Press, London, pp. 175–193.

Chapter 2

An introduction to interdomain user authentication

Contents

This chapter introduces interdomain user authentication and related subject areas, and places this thesis into the context of existing literature and practice. It highlights the contributions of this thesis and presents its structure.

2.1 Elements of SSO

In order to examine the building blocks for an SSO scheme and their origins, one has to examine several interconnected research areas, including secure communication, entity authentication in general, human user authentication in particular, and identity management systems. In the following, we provide a brief overview of each of these areas and discuss how they are brought together in the real world.

2.1.1 Secure communication

In multi-domain computer networks, messages between two endpoints, say A and B, will potentially have to pass through systems that are not under the control of A's or B's administrative domain. Designers, implementers and researchers are faced with the problem of ensuring the correct and sometimes confidential delivery of these messages. Many methods have been devised to protect messages from accidental and malicious modification and interception while they travel through unreliable, untrustworthy or hostile parts of a network.

One class of solutions to this problem (e.g. [32, 114, 115]) is based on symmetric cryptographic techniques, such as the Data Encryption Standard [146]. Symmetric schemes, in general, require A and B to share a common cryptographic key before they can start using the scheme. A seminal paper on some of the key issues involved is that of Popek [170].

A second class of solutions to this problem is based on the use of public key cryptography [69, 74] and example schemes of this type include [143, 205]. This type of cryptography does not require A and B to share a common key; such techniques are also called 'asymmetric' (following Simmons [185]). In contrast to purely symmetric schemes, public key techniques enable A to encrypt a message such that only B can decrypt it, without having to previously agree a common key with B. The message can, of course, itself be a key. Other types of public key cryptosystem support the generation and verification of digital signatures and the establishment of shared secret keys. In the latter case, the key established can then be used to protect subsequent communications using symmetric cryptographic techniques.

Schemes that merely encrypt the exchanged messages, whether symmetric or asymmetric, only preserve the *confidentiality* of the communications, i.e. they prevent a *passive* (i.e. eavesdropping) adversary from reading the contents of the exchanged messages. In many situations it is equally, if not more, important to protect the *integrity* of the communications, i.e. to prevent an *active* adversary from modifying the messages as they are in transit between A and B, without being detected.

2.1.2 Entity authentication and key establishment

Clearly, an active adversary that is in control of the network between A and B can always pretend to be either one to the other [30]. In the face of such an adversary it makes little sense for A and B to establish a key using a protocol that does not provide assurance to A that the key is indeed established with B and assurance to B that the key is indeed established with A. This is because there is always the possibility that the adversary tricks them into establishing the key with itself, rather than the intended communications partner. In order to prevent this type of attack, it is necessary that the protocol provides assurance to A about the identity of B and assurance to B about the identity of A. Schemes that provide this assurance are called 'mutual authentication schemes' and the entities that obtain such assurances are said to 'authenticate each other'. Certain authentication schemes provide assurance to only one of the communication partners (e.g. A) about that identity of the other (e.g. B). Those schemes are called 'unilateral authentication schemes' and, in this case, it is said that 'one entitity (e.g. A) authenticates the other (e.g. B)'.

In many cases it makes sense for A and B to establish a shared secret key at the same time as they authenticate one another (i.e. to perform what is called 'authenticated key establishment'). The established key is then used for the protection of subsequent communications. This is necessary because without this protection the adversary can do whatever it was able to do before authentication (e.g. intercept or modify the communication or hijack the entire session). For this reason, many (but by no means all) entity authentication schemes enable the communication partners to establish a key, typically as a product of a successful authentication event. This key, typically called a 'session key', can be used to protect the integrity and confidentiality of subsequent communications between A and B; the protected communications path that results is said to provide a 'secure channel' between A and B. We use this term throughout the thesis.

A variety of techniques exist to formally verify the correctness of authentication schemes. These include the so-called BAN logic [35], the 'Dolev-Yao' model [70] and a complexity theoretic treatment that is commonly referred to as 'provable security' [17, 18]. The first two approaches led to what is known as the 'formal methods' treatment of authentication schemes [113, 140, 194]; a relatively recent overview of this research area, with many pointers to relevant papers is given in [91]. An interesting comparison between the two schools of thought (formal methods and provable security) is given in [1]. Interestingly, the security of

the Needham-Schroeder protocol (an authentication scheme that appeared in the literature in the late seventies) [148], as modified to the recommendations given in [135, 136], has been the subject of recent analyses using these verification techniques, some 26 years after it was first published (see, for example, [8]).

It is worth noting that each of the above verification techniques (and their derivatives) has its particular set of underlying assumptions, adversarial model and, therefore, limitations. These have been highlighted in [28, 118]. In part II of this thesis we use the setting of provable security in order to define a model and security notions that apply to anonymous credential systems. and show how to construct an SSO scheme based on such systems. Part III of the thesis presents its overall conclusions.

Over the years, many authentication and key establishment protocols have been proposed in the literature (see, for example, [12, 24, 27, 29, 43, 44, 122, 141]). A fairly extensive collection of such schemes and good starting point for the interested reader is the book by Boyd and Mathuria [30].

2.1.3 Human user authentication

The above discussion of authentication and key establishment schemes avoided any assumptions about the nature of the two communicating parties, A and B. In those schemes, A and B could be, for example, network routers, web servers, smartcards, wireless devices or arbitrary combinations thereof. If one of the participants is a human being, an authentication scheme has to take this into account. The primary difference between computers and humans in this respect is the fact that, while computers can accurately store and quickly perform complicated computations on binary data, humans cannot.

The designer of a human/computer authentication scheme has to take into account a number of considerations. These can be divided into those that apply for user-to-computer authentication, and those that apply for the converse.

- Human to computer authentication: As discussed earlier, a human user needs to have some form of authentication credential in order to authenticate himself [64, 112]. Such a credential is typically a piece of information that falls into one of the following

categories [190].

- a piece of information that the user knows, e.g. a password or the answer to a personal question,
- a piece of information that is stored in some hardware that the user possesses, e.g. a smartcard or a similar token that contains a particular cryptographic key, and
- a piece of information that can be derived from some characteristic of the person, e.g. his fingerprint, facial characteristics, or some other biometric.

Some human authentication schemes require the user to use credentials that belong to two or three of the above types. These schemes are known as two-factor and three-factor authentication schemes respectively.

- Computer to human authentication: In this scenario, the user typically wants to authenticate a remote computer (e.g. a web server) using a local machine that communicates with the remote computer over a network. The local machine is typically assumed to be trustworthy. Given that authentication by definition involves verifying a claimed identity, the remote computer is required to have a human-readable and unambiguous identifier. This identifier has to be made known by the local machine to the user, so that the user can decide whether or not it represents the intended (remote) computer. The strength of the resulting computer-to-user authentication depends not only on the security of the scheme, but also on the strength of the computer-to-identifier binding, and, of course, on how informed the user's decision-making is during the authentication process.

The most widely used user-to-computer authentication mechanisms use credentials that the user is required to memorise, typically passwords. This is because these mechanisms are significantly cheaper and easier to deploy than other schemes. Unfortunately, the fact that an attacker can always guess a poorly chosen password makes these systems dependent on human ability and willingness to choose and memorise 'good' passwords. While the exact definition of 'good' may depend on the context or application, a useful guide for many situations is given in [60]. Password-based authentication schemes are not the main focus of this thesis.

Hardware security tokens of various types have been in use for decades. They typically contain a unique piece of information, such as a cryptographic key, that is used in a protocol

executed with a remote entity that requires the user (i.e. the bearer of the token) to authenticate himself. For the protocol to be executed, the token has to communicate with the user's local machine in some way. Different types of token use different channels to communicate with the user's local machine. The Fortezza token from Mykotronx[2.1], for example, occupies a dedicated PC slot, the eToken from Aladdin[2.2] communicates over a Universal Serial Bus (USB) interface [59], some of the Crypto-Box tokens[2.3] communicate via a PC's serial or parallel interface [73, 97], Europay Mastercard Visa (EMV) cards [75, 76, 77, 78] require special card readers, and SecurID tokens from RSA Security[2.4] require the user to manually enter a code into his local machine. Given that in some cases no public information is available, it is hard to say how widely deployed some of these security tokens are. However, with almost one billion users worldwide, the most widely deployed token is almost certainly the Subscriber Identity Module (SIM) of mobile telephones[2.5].

Of course, security tokens and the authentication schemes associated with them are not immune to attacks. As they typically contain a sensitive piece of information, such as a cryptographic key, it is a typical requirement for the token to provide a degree of tamper-resistance. This is necessary because knowledge of this data or manipulation of the token's behaviour may enable an adversary to launch serious attacks on the system. Attacking the token's tamper resistance is not the only type of attack relevant to security tokens. A taxonomy of the various types of attack can be found in [175]. Concrete attacks on the cryptography employed by the SecurID token are documented in [23, 61].

In this thesis we describe possible ways to use existing deployed security tokens and infrastructures in order to build schemes that provide user authentication in a context different to that in which the token was originally deployed. In particular, in chapter 4 we show how to employ the GSM SIM, in chapter 5 we consider the use of EMV cards, and in chapter 6 we discuss the use of PC security hardware known as a 'Trusted Platform Module' (TPM) which shares some of the characteristics of a security token.

The use of biometric identification systems has also been discussed for a number of years, although the robustness of some mechanisms has been criticised (e.g. [172, 199]). Moreover, despite widespread discussion, it is unclear whether any of these techniques have been de-

[2.1] `www.mykotronx.com`
[2.2] `www.aladdin.com`
[2.3] `http://www.dongles.com/ProductPage.htm`
[2.4] `www.rsasecurity.com/securid`
[2.5] In particular, at the time of writing (Jan 4th 2005, 20:45 UTC), the counter on the GSM Association's web page (`www.gsmworld.com`) reported the figure of 990.824.792 mobile phone subscribers worldwide.

ployed on a large scale. Biometrics that have been used in a human authentication context include fingerprints, characteristics of the iris and the retina, and the geometry of the face, the hand and the ear. Biometric methods also include voice recognition, as well as recognition of behavioural characteristics such as keystroke dynamics, walking characteristics and handwritten signatures. The reader interested in biometrics is referred to [5, 206, 145, 204]. In this thesis we do not consider the use of biometrics.

One of the most widespread (remote) computer-to-user authentication schemes is the HTTPS scheme [116, 179], as part of which an TLS/SSL channel [180] is established between a web browser and a web server in the Internet. In this scheme the server's unique identifier is its domain name (which coincides with the first part of the URL [21] of the requested web page), and this should be manually inspected by the user for correctness. Of course, as has been widely discussed, this requirement is ignored by many users; indeed, many users are probably unaware of it [108].

Some human user authentication schemes do not result in the establishment of a key between the local and remote computer. Those schemes do not provide the means to establish a secure channel. Typing a username and a password into a web page, for example, does not result in a secure channel being set up between the browser and the web server. However, it might strengthen the security of an already existing secure channel. The reader interested in a more comprehensive introduction to user authentication is referred to [64, 112].

2.1.4 Identity management

A typical Internet user may have a relationship with multiple service providers. He might adopt a variety of different roles in these relationships. With an online shop, for example, the user adopts the role of a customer; with a governmental service, that of a citizen, and when interacting with his employer's systems, that of an employee. With each role the user may assume a different identity, and in many cases these identities should be kept separate from each other. As the number of such roles increases, so does the burden imposed on the user by the requirement of managing them in a secure and privacy-aware manner.

Identity management systems aim to help the user with this task. Their scope is quite general. In particular, their purpose is to "enable the user to control the nature and amount

of personal information that he releases during his electronic communication" [58]. This may include creation and management of the user's identifiers (i.e. pseudonyms), formulation of privacy policies and preferences, negotiation of policies with service providers, controlled release of location information, etc. The identity management schemes described in [58, 106] are examples of such systems. The business case for identity management is discussed in [187]. Several established infrastructures and standards, including SSO schemes, have been analysed from an identity management perspective (see e.g. [3, 34, 66, 95, 151, 165, 176]).

2.1.5 Single sign-on

A scheme is said to be an SSO scheme if it enables multiple SPs or applications to authenticate a given user, without necessarily requiring the user to authenticate himself more than once. The SSO problem is an old one. The Resource Access Control Facility from IBM, a system that manages user passwords for multiple applications, was first released in 1976[2.6] [189]. In the academic world, researchers have addressed similar problems in the context of 'distributed systems' (see, e.g. [62]). The first version of the Kerberos protocol [192], a system for user authentication to multiple network services, was developed in the 1980s. It is worth noting that the term "Single Sign-On" has been used in the context of operating systems and administration (e.g. [94]) as well as in the context of network services (e.g. [72]). Moreover, the term has been used to refer to the authentication of both static data structures (e.g. user identifiers [128]) and dynamic data structures (such as a payment history [182]). In this thesis we are only concerned with SSO in the context of network services where the authenticated piece of information is a user identifier.

With the growth in user interactions with open systems, SSO and identity management become increasingly related concepts. In fact, SSO schemes often form the basis of identity management systems. For example, the Liberty Alliance specification [130], a platform for permission-based attribute exchange, is based on the Security Assertion Markup Language (SAML) [150], a specification that focuses (among other things) on interdomain SSO. In the literature the terms SSO and Identity Management are sometimes used interchangeably. However, at least in principle, it is not a requirement for an identity management system to provide an SSO service.

[2.6] `www-1.ibm.com/servers/eserver/zseries/zos/racf`

In this thesis we distinguish between identity management and SSO. If an SSO scheme is used as part of a more general identity management system, we regard it as being the part that is responsible for the authentication of the user's various identities (i.e. identifiers) to different services in the network. The term SSO encompasses both closed and open environments. While we do not concern ourselves with identity management in the general sense, it is helpful to keep the relationship between these two concepts in mind.

2.2 Motivation

As the nature of modern digital networks evolves, the construction and analysis of suitable SSO schemes remains an active research area. Today we have reached a point where users have relationships with many service providers; the need for SSO is therefore clear.

Roughly speaking, there are three main motives for SSO.

- **Usability**: This is probably the most obvious advantage of SSO. The user no longer has to maintain a set of authentication credentials for each SP. Moreover, he does not have to do so *securely*.
- **User management:** In a certain type of SSO scheme (see section 3.4.7) a supporting management system can enforce a global policy. Apart from unifying rules and trust relationships among different entities, such a system has the potential to considerably reduce operational costs in a corporate environment. For instance, users can be added to, or removed from, the system by being granted or having revoked a single authentication credential, respectively.
- **Security:** SSO has the potential to increase the overall level of security while also providing usability benefits. From the user perspective, it is arguably easier to securely maintain only one set of authentication credentials, rather than many. However, having only one authentication credential for many services may also pose a risk; if it is compromised, it might enable an adversary to illegitimately access all services. Thus, the credential must be maintained in a secure manner. From the SP perspective, a globally enforceable policy is highly desirable because it has the potential to significantly mitigate the threat of human error and abuse.

2.3 Review of existing literature

This section provides a more detailed overview of the existing literature on the subject of SSO in a networking context.

One could argue that a simplistic form of SSO is achieved if one uses the same password (and possibly user name) with more than one SP. This, however, is not an SSO scheme as such; it is merely a practice that increases user convenience at the cost of security. The dangers of such a practice are discussed by Ives, Walsh and Schneider [105]. Indeed, the existence of this practice provides one of the main motivations for the use of SSO systems. This issue is not discussed further here.

One of the first uses of the term SSO in the sense we mean it here can be found in a paper by Grubb and Carter [94]. The authors discuss various approaches to the provision of SSO in a closed environment, and document their experiences in deploying an enterprise-wide SSO system. Volchkov outlines the design decisions taken for the deployment of an SSO scheme for a Swiss bank [202]. Josephson, Sirer and Schneider describe an SSO scheme where, in order to distribute trust, multiple servers are required in order to provide the authentication service [109]. More recently, Linden and Vilpola conducted an empirical study of a deployed SSO system and argue that such systems should also offer a 'single logout' mechanism [134]. The schemes proposed in this thesis are designed for open environments and take into account the 'single logout' requirement.

Well-known SSO schemes include Kerberos, the SAML and Liberty specifications, and Microsoft Passport. Kerberos [99, 192] has been implemented and deployed in a number of applications and scenarios. The Kerberos system has been revised a number of times. An overview of its early evolution is given by Kohl [119]. Bellovin identifies certain limitations of the scheme [20] and Ganesan proposes an extension that uses public key cryptography [85]. Kerberos is discussed in more detail in section 3.5.1.

The Liberty Alliance [128] and the Security Assertion Markup Language (SAML) [150] standards define a set of protocols that provide a web-based SSO service that enable a user to log into multiple web sites using a single authentication credential. Similarly, Microsoft Passport [144] is a web-based SSO scheme that has been in use for over five years. More detailed descriptions of these schemes are provided in sections 3.5.2 and 3.5.3 respectively.

Several papers discuss various aspects of the SAML and Liberty Alliance specifications. In [165, 166], for example, Pfitzmann examines privacy policies for the Liberty Alliance specifications and shows that the specification of such policies is a non-trivial task. In [167], Pfitzmann and Waidner discuss, among other things, privacy aspects of Liberty, SAML and Microsoft Passport with a focus on attribute exchange. In [168] the same authors discuss certain 'profiles' (i.e. modes of operation) of Liberty. In [92, 93] Gross and Pfitzmann analyse the security of another set of profiles of SAML-based web SSO protocols and give a number of recommendations to implementors. Hansen, Skriver and Nielson use formal methods to analyse certain instantiations of the SAML SSO protocol [96]. Jeong, Shin, Shin and Moon describe an implementation of a SAML-based SSO scheme in [107]. Li, Ge, Wo and Ma describe the implementation of an SSO scheme that also makes use of SAML in [126].

The ideas on which Microsoft Passport (and, to a lesser extent, SAML and Liberty) are based, namely the exchange of authentication information using HTTP cookies, is discussed by Samar [181]. Kormann and Rubin identify a number of attacks on Microsoft Passport in [120]. Oppliger discusses Microsoft Passport from an identity management viewpoint in [151].

Other SSO schemes that have been proposed in the literature include those of Satoh and Itoh [182], and Ishitani, Almeida, and Meira Jr. [103]. Moreover, a number of commercial SSO products that use proprietary protocols have been developed (see, e.g. [189, 144]).

In order to establish a basis on which the above schemes can be compared, in chapter 3 we establish a taxonomy of SSO systems. The taxonomy enables us to put into perspective existing work, and to discuss our contributions in a common context. Part I of this thesis proposes a number of new SSO schemes that are either based on existing technology or have properties that other existing systems do not possess. Wherever this is necessary, we provide a discussion of relevant work and discuss our schemes within the context of the taxonomy. Some of the proposed schemes, e.g. those discussed in chapters 4 and 5, could be implemented as enhancements or extensions of existing schemes. Although we do not discuss the details of how this could be done, the modular nature of the Liberty Alliance specifications, for example, may allow for such extensions to be specified in a straightforward manner.

Part II of this thesis considers anonymous credential systems with an emphasis on aspects

relevant to their use in an SSO scenario. Chapter 8 provides a separate literature overview of that area. In this part also two methods to build an SSO scheme using an anonymous credential system are discussed. Such an SSO scheme provides stronger privacy and security properties than all existing schemes. The following section provides a more detailed description of the contributions of the thesis.

2.4 Overall structure and summary of contributions

This section briefly outlines the structure of this thesis and highlights its main contributions. The remainder of the thesis is divided into three parts. In general terms, part I is concerned with pragmatic 'real-world' SSO schemes that are based on well-established and available infrastructures. Part II is concerned with anonymous credential systems and with combining anonymous credential systems and SSO. Finally, part III presents the overall conclusions of the thesis. We next highlight the contributions of each part in more detail.

Part I: A number of different interdomain user authentication systems have been deployed and are in use today. In chapter 3 we establish a taxonomy of such systems and discuss their strengths and weaknesses. Within our taxonomy, we identify four different categories of SSO scheme. The taxonomy is subsequently used as a basis to put into context the novel and pragmatic SSO schemes that we propose. Four such schemes are proposed, as follows.

- **Chapter 4**: We describe the construction of an SSO scheme that is based on the infrastructure of 2nd and 3rd-generation cellular (mobile) telephony. In particular, the scheme uses a subscriber's Security Identity Module (SIM) for interdomain user authentication. With more than one billion users worldwide, it is clearly advantageous to be able to use the SIM in the context of SSO. The scheme proposed in this chapter does not require changes either in the SIM or in other strategic network components of the system. We also consider certain relevant attacks and countermeasures.
- **Chapter 5**: We describe the construction of an SSO scheme that is based on credit/debit smart cards that conform to the Europay, Mastercard, Visa (EMV) specifications. These cards are being deployed in a number of countries and are supported by a worldwide infrastructure. The SSO scheme reuses the cards and the established key management infrastructure in order to facilitate intradomain user authentication.

While the scheme requires certain changes in the card itself, it does not require changes in any other equipment. We consider certain attacks and countermeasures and discuss relevant issues.

- **Chapter 6**: We describe the construction of two SSO schemes that are based on trusted computing, i.e. on platforms that conform to the Trusted Computing Group specification. Such platforms come with a cryptographic co-processor that can perform certain functions and have been on the market for more than a year. The reason why we describe two schemes is that they belong to two different categories of the taxonomy of chapter 3. Both are potentially useful in an appropriate environment. We discuss issues that arise in the context of the proposed schemes, such as privacy, complexity and cross-platform mobility.

- **Chapter 7**: While the previous three SSO schemes are based on tamper-resistant hardware modules, in this chapter we describe the construction of an SSO scheme that is not. Its distinctive feature is that it is suitable for use from an untrusted network access device, i.e. a device to which no long-term secrets should be made available. The scheme overlays a one-time authentication mechanism over the legacy mechanisms of individual service providers. We also describe an open-source implementation of the scheme that works as a web proxy. The implementation performs additional functions such as the transparent handling of secure channels and content filtering.

Part II: As the need to preserve one's privacy continues to gain importance in the digital world, it is important to enhance user authentication schemes with properties that enable users to remain anonymous (yet authenticated). In the second part of the thesis, anonymous credential systems are identified as a tool that can be used to help achieve this goal. This part also shows how to use an anonymous credential system in order to facilitate what we call 'privacy-aware single sign-on' in an open environment. In particular,

- **Chapter 8** provides a literature overview and a short informal introduction to the topic of anonymous credential systems and pseudonym systems. It also points out that there exist inherent limits to the degree of privacy that can be obtained in the context of anonymous credential systems, and describes a series of timing attacks that exploit these limits. This chapter further investigates the possibility of imposing randomised waiting times between the issuing and showing of credentials, and discusses

the tradeoffs involved. Some heuristics that help mitigate exposure to the attacks are proposed.

- **Chapter 9** presents a formal model that captures relevant security and privacy notions for anonymous credential systems. The model essentially represents a complexity theoretic treatment of such systems and exists within the setting of provable security. It provides insight into the relationships between different security and privacy notions.
- **Chapter 10** presents a peer-to-peer scheme that provides a level of protection against the timing attacks described in chapter 8. The approach differs from that of in chapter 8, and also provides real-time feedback to users about the level of privacy they enjoy.
- **Chapter 11** introduces the notion of a privacy-aware SSO scheme and provides a brief discussion of why existing systems cannot be classified as such.
- **Chapter 12** presents two ways to construct a privacy-aware SSO scheme using an anonymous credential system. It further discusses certain issues that arise and the tradeoffs involved between usability, security and privacy.

Part III: This part presents the overall conclusions of the thesis. In particular,

- **Chapter 13** presents the conclusions of the thesis and gives directions for further research.

Part I

Interdomain User Authentication

Chapter 3

A taxonomy of distributed authentication architectures

Contents

The aim of this chapter is to introduce some of the existing interdomain user authentication systems and to develop a taxonomy of these schemes. This taxonomy serves as a basis to put into context the schemes that are proposed in the remaining chapters in this part of the thesis. Much of the material in this chapter has previously been published in [157].

3.1 Introduction

A number of different interdomain user authentication schemes have been proposed. As each of these has been designed to meet the requirements of a particular operational environment, different systems typically rely on different assumptions and adopt different architectures. By considering the various possible architectures, we identify four generic types of real-world user authentication scheme. We discuss their relative strengths and weaknesses, and use our taxonomy as a basis to put into context the schemes that are introduced in the remainder of this part of the thesis.

An interdomain user authentication system, also called an SSO scheme, is defined as a scheme that enables the user to authenticate himself to more than one service provider (SP) using only a single authentication credential. As authentication implies identification, a real-world system has to incorporate the life cycle management of the identifiers by which a user is known to the SPs he is registered with. These identifiers can take various forms (e.g. e-mail addresses, names, or sequence numbers). In order to maintain generality, in this chapter we make no assumptions about the type of name used, and simply treat names as finite sequences of bits. In the remainder of this chapter we refer to them as 'SSO identities'.

In the next section we give a high level overview of the operation of an SSO scheme. Although concrete schemes differ in many, often fundamental, aspects, the description here is generic; whilst it omits the technical details, it applies to virtually all existing such schemes. We then present the taxonomy in section 3.3. Section 3.4 discusses the relative strengths and weaknesses of SSO schemes in each of the four classes, and section 3.5 presents some existing, real-world example schemes in the context of the taxonomy. Finally, section 3.6 gives a short summary of this chapter.

3.2 How SSO works

Virtually all existing SSO schemes depend on the notion of authentication *sessions*. The system information flow, depicted in Figure 3.1, is as follows. First, in order to start a session, the user needs to authenticate himself, using his single authentication credential, to a special entity which we call the 'Authentication Service' (AS). If this 'initial' authentication

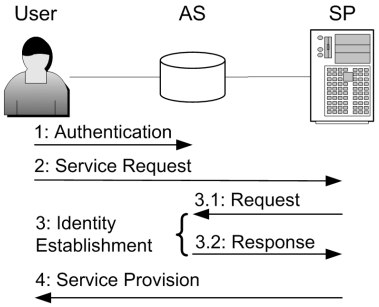

Figure 3.1: Information flow of a generic SSO system.

is successful, the session is started (step 1 in the Figure).

For as long as the session lasts, and whenever a service is requested (step 2), the AS provides the following service to the user: it *automatically* logs him into the SPs he uses (step 3). How exactly this is done differs from scheme to scheme. In all schemes, however, it involves the AS, and possibly the user, executing a protocol with the SP; we call this the 'SSO protocol'. The objective of this protocol is to identify and authenticate the user to the SP in a manner that does not (necessarily) require further manual interaction from the user. Finally, assuming that step 3 was successful, the service is provided in step 4.

At some later point in time the authentication session will be terminated. The reasons for this termination will again vary from system to system. Sessions are, of course, subject to the policies of the AS, the SP and the user, and reasons for termination may include events such as extended periods of inactivity, a maximum number of logins performed, or a simple time limit. After a session has been terminated, the user has to authenticate himself again to the AS in order to start a new session.

3.3 The taxonomy

We distinguish between two main types of user authentication schemes for distributed systems. In the first type of scheme, which we call 'pseudo-SSO', the AS achieves automatic user authentication simply by executing whatever authentication mechanisms are in place at the different SPs, as necessary. The key distinguishing feature of this type of system is that, during the session, a separate user authentication occurs every time the user is logged into an SP. The AS manages the SP-specific authentication credentials, which constitute

the SSO identities in this case. Since these SSO identities are SP-specific, the SSO Identity/SP relationship is $n : 1$; that is, any given SSO identity corresponds to exactly one SP, and a user may, in principle, have multiple SSO identities for a single SP. This type of SSO scheme is illustrated in Figure 3.2.

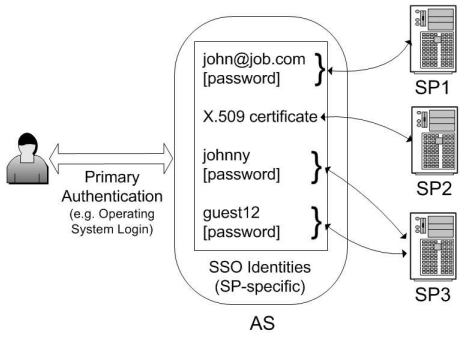

Figure 3.2: Pseudo-SSO

Pseudo-SSO is fundamentally different from the second type of distributed user authentication scheme, which we call 'true SSO'. In a true SSO scheme all SPs that are part of the SSO system are required to have an established relationship with the AS. This relationship requires a level of trust that is typically supported by a contractual arrangement. There would also typically need to be a supporting infrastructure to enable secure communications between the AS and the SPs. The key distinguishing feature of true SSO is that, from the SP viewpoint, a trusted third party, namely the AS, provides the authentication service; the SP is notified of the authentication status of the user via so-called *authentication assertions*. These are statements that contain the user's SSO identity and his/her authentication status at the AS. Note that the transport of these authentication assertions will itself need to be secured by some means. Also, the format of SSO identities depends on the system design, but would typically be uniform throughout the whole system.

In contrast to pseudo-SSO, under true SSO the SSO identity/SP relationship can be $n : m$. That is, if supported by the scheme, not only can the user potentially choose from a 'pool of identities' for any given SP, but the same SSO identity could, if the user wishes, be used with multiple SPs. This enables the assignment of specific *roles* to SSO identities (which then act as 'role pseudonyms' as defined in [164]). The operation of a true SSO scheme is depicted in Figure 3.3.

User authentication schemes for distributed systems can be further categorised based on the location of the AS; specifically, the AS can either be local to the user platform, or offered

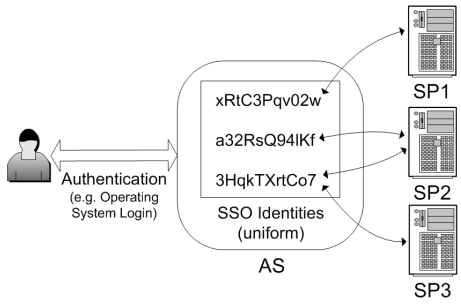

Figure 3.3: True SSO

as a service by an external entity, which we refer to as the 'SSO proxy'. We thus arrive at the following four main categories of user authentication schemes for distributed systems:

- Local pseudo-SSO systems,
- Proxy-based pseudo-SSO systems,
- Local true SSO systems, and
- Proxy-based true SSO systems.

We next consider each of these four types of scheme in a little more detail.

3.3.1 Local pseudo-SSO

In a local pseudo-SSO system, the AS is resident within the user machine. The AS maintains a database of authentication credentials for all the SPs that the user has a relationship with. This database may be encrypted in order to prevent an adversary with temporary access to the machine from stealing the credentials. The user authenticates himself to the AS at the beginning of a session. From that point on, the AS automatically executes SP-specific user authentication mechanisms whenever needed, making use of the appropriate authentication credentials. Of course, encrypted credentials have to be decrypted first; therefore, the decryption key has to be made available to the machine, at least during the session. A key property of this architecture is that the machine needs to have access to the user's long-term authentication credentials. Thus, the machine is required to be sufficiently trustworthy for this purpose.

3.3.2 Proxy-based pseudo-SSO

In a proxy-based pseudo-SSO architecture, the AS resides on an external server. This external server, called a proxy, has to have access to the user's credentials in unencrypted form, and hence must be trusted for this purpose. The initial authentication occurs between the user and the proxy at the beginning of a session (and possibly thereafter if the proxy wishes to perform a re-authentication). Subsequent user authentication at SPs is redirected to, or intercepted by, the proxy, which automatically executes the SP-specific authentication mechanism on the user's behalf. A key property of this architecture is that, as authentication occurs directly between the proxy and the SPs, the local machine is not necessarily required to have access to the user's SP-specific credentials. The scheme described in chapter 7 belongs to this category.

3.3.3 Local true SSO

Under true SSO, the AS authenticates the user, and subsequently conveys authentication assertions to SPs whenever necessary. When a trusted component within the user system takes the role of the AS, the resulting architecture falls into the category of local true SSO systems. A trust relationship between the local AS and the relying SPs has to exist. This trust relationship has to be supported by an appropriate security infrastructure. Moreover, since, in this setting, the AS resides in the user's machine, it is under the physical control of the user. That is, a malicious user could modify the local AS to make it falsely assert to an SP that a different user has been authenticated. Therefore, mechanisms must be in place that guarantee the AS's integrity. The schemes described in chapters 5 and 6 belong to this category.

3.3.4 Proxy-based true SSO

Finally, in a proxy-based true SSO architecture, an external server takes the role of the AS. This external server acts as a broker between users and SPs. A user first authenticates himself to the AS. Using an appropriate SSO protocol, he is subsequently automatically logged into those SPs that maintain a trust relationship with the AS. The schemes described in section 3.5.2 and 3.5.3 of this chapter and in Chapter 4 belong to this category.

3.4 Properties of user authentication schemes for distributed systems

In this section, properties that are inherent to SSO schemes in the four classes of the taxonomy are presented. In order to maintain generality, the discussion is at a high level of abstraction. An interesting report on a real world deployment of an SSO system, where many of the design decisions were based on the properties described below, can be found in [94].

3.4.1 Privacy protection

Privacy protection is of particular importance in open environments [207]. With respect to SSO systems, privacy concerns arise in a number of situations. If, for example, SSO identities contain personally identifying information, it may be possible for colluding SPs to correlate distinct identities of the same user without his consent, thereby aggregating their respective information about the user into a single profile. Guaranteeing SSO identity pseudonymity, in the sense that the identity does not include any personally identifying information [164], is therefore a potentially desirable property.

Of course, such unlinkability may not always be possible in practice, e.g. if the SP is required to have access to the user's real name and address for the delivery of physical goods. Moreover, depending on the legal jurisdiction applying to the SP, aggregation of personally identifying information may be severely restricted by privacy legislation. Nevertheless, guaranteeing pseudonymity by technical means may be of genuine significance in cases where SPs are providing goods and services that are delivered electronically, and in domains where legal protection for user privacy is inadequate, or even nonexistent.

In the remainder of this section we focus on the case where users are known to different SPs under different *pseudonyms*. A privacy-protecting SSO scheme should, in this case, prevent SPs working out which pseudonyms belong to the same user — even if they collude with each other and the AS provider. An SSO scheme that protects the user's privacy in this sense is said to offer *unlinkability of pseudonyms*[3.1] [164, 191]. It turns out that unlinkability is not as easy to achieve as it may seem at first glance. In fact, at the time of writing and

[3.1] Unlinkability of the user's pseudonyms is examined in greater detail in part II of this thesis.

to the best of the author's knowledge, no existing real-world SSO scheme has managed to achieve it.

In order for pseudonyms to be unlinkable, they should not contain any information that allows them to be linked to each other. Under pseudo-SSO, the SP will typically select the type of user identifier (i.e. pseudonym) that has to be used; a user interested in privacy must select his pseudonyms in a way that obscures the fact that they all belong to him. Apart from this being a task that many users may find impractical, the system neither inherently aids in, nor prevents, linking of pseudonyms; in this sense unlinkability is not within the scope of such systems.

A system that is designed to offer unlinkability of pseudonyms necessarily needs to provide a suitable mechanism for users and SPs to establish these pseudonyms. A true SSO scheme has the potential to offer such a mechanism as an intrinsic part of its operation. This is because, since the applicability of a true SSO scheme's protocols is system-wide (i.e. *all* SPs are required to support the *same* protocols), SPs can be required to support a suitable pseudonym establishment mechanism. Unfortunately, providing such a mechanism is not sufficient; it is also required that subsequent use of the scheme (executions of the SSO protocol in particular) does not compromise the unlinkability of pseudonyms. The second and third parts of this thesis are devoted to this subject.

3.4.2 Anonymous network access

Whether or not otherwise unlinkable SSO identities remain so when an adversary has access to network address information depends on the wider context within which the SSO scheme is deployed. If, for example, the lower layer protocols do not provide an anonymous user-to-SP channel, the adversary can easily correlate SSO identities using information found in packet headers or traffic analysis [7]. Unfortunately, the user's network address (which *is* typically included in packet headers) also reveals the identity and geographical location of his/her network access provider. The provider can then help link the network address (and thereby the SSO identity) to the user's real name and address.

These privacy issues are addressed by so-called 'anonymous network access' schemes. In these schemes, all traffic between the user and the SPs is physically routed through an

externally operated server, known as an 'anonymising proxy'. This proxy replaces the user's real network address (and perhaps other identifying information) with its own. The level of anonymity achieved depends, of course, on the number of users in the system [164]. Examples of such anonymity services are the Anonymizer[3.2] and Freedom WebSecure[3.3] from zerøknowledge. While single-proxy schemes provide only a limited level of resistance against traffic analysis, the proxy operator has to be fully trusted to provide the anonymising service. So-called mix networks [45, 89] address these issues by distributing the proxy functionality over several servers. A real-world example of such a system is JAP[3.4].

Anonymising services can be employed in conjunction with SSO systems in order to increase the level of SSO identity unlinkability. In fact, SSO-proxies could be augmented with the functionality of an anonymising proxy. Of course all traffic would have to be physically routed through the proxy, but the user would not have to trust an additional entity. In the case of proxy-based pseudo-SSO, the architecture might be completely transparent to SPs: they would not even need to be aware of the proxy's existence. The scheme described in chapter 7 is based on a proxy that, among other things, hides the user's network address from SPs.

3.4.3 User mobility

Proxy-based architectures inherently support user mobility; users can authenticate themselves to the AS from anywhere in the network (except, of course, if the authentication method itself imposes restrictions on this). We thus concentrate on how user mobility can be supported in local schemes.

In local pseudo-SSO systems, user mobility can be supported if the credential database is held on an external server: initial authentication between user and server would occur once at the beginning of a session, and credentials would then be downloaded to the local machine as needed. The degree of trust required in the server in such a setting varies according to whether or not credentials are stored in an encrypted form. Such a setting does not constitute proxy-based SSO; it is the local machine that accesses the credentials in

[3.2] www.anonymizer.com
[3.3] www.freedom.net/products/websecure
[3.4] anon.inf.tu-dresden.de

the clear and executes the SSO protocol. Novell's SecureLogin[3.5], Passlogix' V-GO[3.6] and Protocom's SecureLogin[3.7] products are examples of local pseudo-SSO systems with support for user mobility. One can also regard automatic form-fillers as products with pseudo-SSO functionality. Examples include the automatic form completion functions of popular web browsers and Novell's DigitalMe[3.8] service.

Whether or not local true SSO schemes support mobility depends on the mechanism that is used to provide assurance to the SP regarding the integrity of the AS. The local true SSO scheme described in chapter 5, for example, supports user mobility since the hardware token on which it is based is itself mobile. On the other hand, in the scheme described in chapter 6, user mobility is not supported per se, because the scheme involves a platform-specific hardware module.

3.4.4 Use in an untrusted environment

Sometimes users wish to access SPs from untrusted or hostile environments such as Internet cafés or other types of public or shared terminals. In these situations it is undesirable if the untrusted machine ever has access to long-term authentication credentials that will allow it to later successfully impersonate the user.

In such scenarios, proxy-based SSO schemes are potentially useful. Of course, the initial authentication between user and proxy must have the property that observation of one authentication exchange does not enable subsequent impersonation of the user. This requirement can be met by using suitable challenge/response protocols, such as a one-time password scheme (see, for example, [142]) to initially authenticate the user. Building on this, and assuming that all network traffic between the user and the SPs is physically routed through it, the proxy may also provide an additional privacy protection service by 'stripping off' all personal data before it reaches the untrusted machine. The scheme proposed in chapter 7 is of this form.

[3.5] www.novell.com/products/securelogin
[3.6] www.passlogix.com/sso
[3.7] www.protocom.cc
[3.8] www.digitalme.com

3.4.5 Deployment and maintenance costs

Generally speaking, it is far less costly to deploy pseudo-SSO than true SSO schemes, because they do not require any common security infrastructure; existing SPs may not need to change at all. On the other hand, if an SP *does* change its user authentication mechanisms after deployment, this has to be reflected in the AS. This increases the maintenance costs of pseudo-SSO schemes, especially in dynamically changing environments.

The situation is reversed in true SSO schemes. Deployment of true SSO systems requires a potentially costly, system-wide infrastructure. This infrastructure must both enable SP-AS secure communications (e.g. using a Public Key Infrastructure), and provide the management/legal framework which might include service level and liability transfer agreements. Once the infrastructure is in place, however, maintenance costs are likely to be low, since changes in the user authentication interface occur only between the user and the AS.

3.4.6 Running costs

The running costs of local SSO schemes are likely to be lower than those of proxy-based systems. This is because local SSO schemes do not require the continuous online presence of a server.

Proxy-based systems depend, of course, on the security and availability of the external proxy server. Moreover, the proxy constitutes a single point of failure in the system. As such, it must be protected against service denial attacks. Additional communications costs might be incurred if all traffic is physically routed through the proxy.

3.4.7 Trust relationships

Regardless of the type of a scheme, a dishonest AS can typically impersonate any registered user at every SP at will. This limitation exists in all SSO schemes that the author is aware of, apart from the scheme described in part II of this thesis. Consequently, both users and SPs have to trust the AS. Local systems, such as the one described chapter 6, have the advantage that the user does not have to trust an entity that is under external control,

although he will still need to trust the AS manufacturer.

It is important to note that, despite the universal need for user trust in the AS, there remain differences in the nature of the trust relationships involved in true and pseudo-SSO schemes. In the case of a true SSO scheme, the common security infrastructure allows for the trust relationships between the SPs and the AS to be precisely described and regulated by policies, service level and liability transfer agreements, and other non-technical means. Furthermore, if the user's authentication credentials are compromised, the service can be disabled centrally.

Under pseudo-SSO the trust relationships are more diffuse. SPs may not even be aware that the scheme is in place. Credential databases may be encrypted and replicated at several servers. The trust relationship between the user, any servers and the pseudo-SSO component depends on the implementation details of the scheme. The trust relationship between AS and the SPs may be different for each SP, and may change whenever individual SPs modify their authentication interfaces.

3.4.8 Conflict resolution and lawful access

In the event of a dispute or a lawful investigation, the operator of the proxy in a proxy-based scheme may provide evidence of events, as a trusted third party, by keeping logs of authentication events. The situation is likely to be better defined in a true SSO system, as in such systems there is necessarily a well-defined relationship between the AS and the SPs.

Local SSO schemes are much less likely to be useful in this sense, since evidence can be modified or deleted. However, if the AS in a local true SSO system incorporates physical protection measures that enable it to be trusted by third parties, then locally stored event logs may still possess evidential value. The scheme described in chapter 6 can be enhanced with this property.

3.4.9 Open versus closed environments

The issue of privacy protection (which includes anonymous network access and use in untrusted environments) is usually deemed to be of less importance in closed environments.

Thus, the main focus for SSO for closed systems is likely to be the deployment, running and maintenance costs. Since these are less for pseudo-SSO systems, especially in relatively stable environments, 'enterprise' pseudo-SSO solutions promise a rapid and concrete return on investment. Architectures of such systems are examined in [65], and examples of real-world implementations include Computer Associates' eTrust Single Sign-On[3.9], cafesoft's Cams[3.10], Entegrity's AssureAccess[3.11], Entrust's GetAccess[3.12]and Evidian's AccessMaster[3.13].

However, the need for privacy protection in open environments may well outweigh the deployment cost of true SSO schemes. Thus it seems likely that true SSO systems will eventually be required in open environments such as the Internet. Some well-known systems designed for open environments are discussed in the next section.

3.5 Some examples of SSO schemes

Most existing SSO schemes are either local pseudo-SSO or proxy-based true SSO schemes. Examples of the former class have been mentioned in section 3.4.3. This section provides more detailed descriptions of three well-known SSO schemes that belong to the latter class, namely Kerberos, the Liberty Alliance specifications, and Microsoft Passport.

3.5.1 Kerberos

Kerberos is perhaps the first interdomain 'network authentication system' [99, 192]. A single security domain, or *realm*, consists of a set of users, an Authentication Server, a 'Ticket Granting Server' and a set of relying SPs. The Authentication Server and Ticket Granting Server can be combined into a single entity called the 'Kerberos server'. In terms of the generic system described in section 3.2, the Kerberos server constitutes the AS. The security infrastructure of Kerberos relies solely on symmetric cryptography; every user and every SP share a long-term secret key with the AS. All secret keys are used to perform an encryption operation, which is implicitly assumed to provide integrity protection (in addition to providing confidentiality).

[3.9] www3.ca.com/Solutions/Product.asp?ID=166
[3.10] cafesoft.com/products/cams/camsOverview.html
[3.11] www.entegrity.com/products/aa/aa.shtml
[3.12] www.entrust.com/getaccess/index.htm
[3.13] www.evidian.com/accessmaster

A simplified description of user authentication under Kerberos follows. The protocol is executed whenever the uses wishes to log into an SP of the realm.

1. If the user already possesses a valid 'Ticket Granting Ticket' (TGT) from a previous protocol run, this step is omitted. Otherwise, the user requests a fresh TGT from the AS. The AS replies with a message that contains a fresh TGT and a 'session key' which will be used to construct an 'authenticator', i.e. a data structure which is encrypted under the session key and contains elements that protect against replay attacks. This session key is encrypted under the long-term key the user shares with the AS (or a key derived from it). The user decrypts the session key using his long-term key.
2. The user sends a message to the AS that contains the TGT, an authenticator (encrypted under the aforementioned session key), and the identifier of the SP he wishes to access. The AS checks the validity of the received message. If the check is not satisfied, (if, for example, the TGT has expired) authentication fails. Otherwise the user is now deemed authenticated at the AS.
3. The AS replies with a message that contains a 'Service Granting Ticket' (SGT), a data structure encrypted using the key shared by the AS and the SP in question, and a second session key which is encrypted under the session key of step 1.
4. The user now constructs a message containing the SGT and an authenticator encrypted under the second session key. This message, if constructed correctly, demonstrates the user's ability to decrypt the second session key. Although it is constructed by the user, it can be regarded as an authentication assertion. The user sends it to the SP which decrypts it and, if valid, logs the user in.

The tickets (TGT and SGT) are encrypted data structures that contain the user identifier and network address, the server identifier, session keys and expiry timestamps. The SSO effect is achieved by the fact that the user does not need to re-use his long-term key while the TGT remains valid.

Interdomain user authentication (i.e. across multiple realms) is achieved by setting up the required relationships and symmetric keys between the Kerberos servers. There is no restriction as to the type (web, FTP, etc.) of SPs that may rely on Kerberos for user authentication, as long as they follow the protocol.

Since the same user identifier is used with every SP, the SSO identity/SP relationship under Kerberos is $1 : n$. Thus, unlinkability of SSO identities is not an issue. It is interesting to observe that even distinct Kerberos accounts of a given user are still linkable, as tickets bind them to the user's network address. Since the authentication mechanism is based on a long-term secret key, user mobility can be supported (if the key is derived from a password, for example) but it is not suitable for use in an untrusted environment. Other limitations of the protocol have been documented in [20].

3.5.2 The Liberty Alliance

The Liberty Alliance[3.14], a consortium of over 140 companies, recently developed a set of open specifications for web-based SSO [130, 129, 131, 133]. In Liberty terminology [128], the AS and the user are called the 'Identity Provider' and 'Principal' respectively. The specifications use the Security Assertions Markup Language (SAML), a platform-independent framework for exchanging authentication and authorisation information[3.15] [150]. Interestingly, SAML itself can be used for SSO in a number of different modes of operation. One such mode is analysed by Gross [92].

Liberty is based on the notion of 'trust circles' which are formed by trusted ASs and sets of relying SPs. The AS/SP trust relationship has to be supported by contractual agreements outside the scope of the specifications. According to the specifications, users first authenticate themselves to the AS, which subsequently conveys authentication assertions to the relying SPs. The assertions contain 'name identifiers' that allow SPs to differentiate between users. For any given user, the AS has to use a distinct identifier with each SP in the trust circle. The SSO identity/SP relationship is therefore $1 : 1$. Furthermore, name identifiers "must be constructed using pseudo-random values that have no discernible correspondence with the Principal's identifier (e.g. username) at the Identity Provider [AS]" [133, p.12]; SSO identities are therefore potentially unlinkable.

This unlinkability, however, can be compromised in a number of ways. Firstly, as the AS knows all the user identifiers, SPs could collude with the AS to link the pseudonyms of a user. Secondly, SPs may be able to correlate SSO identities based on the user network addresses. As discussed in section 3.4.2, this issue can be addressed by using an anonymous

[3.14] `www.projectliberty.org`
[3.15] `www.oasis-open.org/committees/security`

network access scheme. Thirdly, profile information that individual SPs may maintain (such as shopping habits, telephone numbers or credit card details) can also be used to link identifiers. Although this last point lies outside the scope of a user authentication scheme, the specifications acknowledge that, for the time being, "the only protection is for Principals to be cautious when they choose service providers and understand their privacy policies" [131, p.70]. An independent assessment of the specifications with respect to privacy appears in [166].

The Liberty specifications are independent of the specific user authentication mechanism (or mechanisms) used by the AS; the details of the particular method that has been employed by the AS to authenticate a user are explicitly stated in the authentication assertions [129]. This means that, if a suitable user authentication mechanism is in place, user mobility or even use in an untrusted environment can be supported by a Liberty-compliant AS.

The Liberty Protocols and Schema Specification [133] defines generic requirements for the protocols for conveying assertion requests and responses between parties. Concrete protocol bindings are only specified in the context of a Liberty *profile*. All currently specified profiles rely on the Secure Socket Layer (SSL) or the Transport Layer Security (TLS) [180] protocol to provide secure channels between parties. Hence, a Public Key Infrastructure (PKI) must be in place. A separate, or at least a more sophisticated, PKI may be required if a profile is used that requires assertions to be digitally signed. Authentication assertions sent from the AS to the SP are routed through the user browser via web redirects; in the 'browser/POST' profile, for example, assertions are sent within an HTTP form, while in the 'browser/artifact' profile an 'artifact' is encoded in the URL that the SP can later resolve into an assertion [131].

3.5.3 Microsoft Passport

Microsoft Passport[3.16] is a web-based SSO service that has been offered by Microsoft since 1999. In Passport, messages are conveyed via 'cookies', an approach discussed in [181]. The passport server acts as the AS. Users register with it by supplying a valid e-mail address and a password (or, if they register from a mobile phone, their phone number and a Personal Identification Number). Additional profile information, such as address, date of birth and

[3.16] `www.passport.com`

credit card details, may also be stored in their passport accounts. Every account is uniquely identified by a 64-bit number called the 'Passport User ID' (PUID). SPs that wish to join the scheme need to sign a contractual agreement with Microsoft (which involves a yearly provisioning fee of $10,000 [144]), implement a special component in their web server software, and share a secret key with the AS. Since SSL/TLS channels are required between the user and the passport server (and optionally between user and SP), an appropriate PKI must also be in place.

User authentication is achieved using the following protocol, which is executed whenever the user wishes to log in to an SP.

1. The user's browser is redirected to the AS.
2. The AS tries to retrieve a 'Ticket Granting Cookie' (TGC) from the browser's cookie cache. If one is found, it decrypts successfully, and is valid, the user is deemed authenticated and the rest of this step is omitted. Otherwise, the AS requests the user to authenticate himself. Assuming successful authentication, the AS saves a fresh TGC in the browser's cookie cache. This cookie is encrypted under a 'master key' only known to the AS. Its function is similar to the TGT of Kerberos; there is, however, no 'authenticator' in Passport — replaying a stolen TGC results in successful impersonation.
3. The AS saves a set of cookies in the browser's cookie cache which include the user's PUID and other profile information that the user has agreed to share at the time of initial registration. This cookie set is encrypted under the secret key shared between the AS and the SP in question. The encrypted cookie set plays a role similar to that of the Kerberos SGT, and acts as an authentication assertion.
4. The user's browser is redirected back to the desired SP, which reads and decrypts the aforementioned cookie set and, if satisfied, logs in the user.

The SSO effect is achieved by the fact that, as long as the TGC remains valid, the user does not need to re-authenticate (in step 2) in subsequent protocol runs. As the authentication method is password-based, user mobility is supported, but the scheme is not suitable for use in an untrusted environment. Passport users, like Kerberos users, use a single identifier (the PUID) with every SP. The SSO identity/SP relationship is therefore $1 : n$, and unlinkability is not an issue.

According to the Liberty Alliance [127], Passport will be based on Kerberos in the future.

3.6 Summary

This chapter has presented an abstract taxonomy of user authentication schemes for distributed systems, and an analysis has been given of the properties of the four main types that were identified. The characteristics of these four types of scheme are summarised in Table 3.1. We have also presented some real-world examples of schemes in the light of the taxonomy.

Table 3.1: Properties of SSO systems.

	Local pseudo-SSO	*Proxy-based pseudo-SSO*	*Local true-SSO*	*Proxy-based true SSO*
Pseudonymity and Unlinkability	cannot be guaranteed	cannot be guaranteed	can be guaranteed	can be guaranteed
Anonymous Network Access	needs additional services	can be integrated	needs additional services	can be integrated
Support for User Mobility	needs additional services	under suitable authentication method	needs additional services	under suitable authentication method
Use in Untrusted Environment	not supported	under suitable authentication method	not supported	under suitable authentication method
Deployment Costs	low	low	high	high
Maintenance Costs	potentially high	potentially high	low	low
Running Costs	low	high	low	high
Trust Relationships	diffuse and changing	diffuse and changing	concrete and consistent	concrete and consistent

It is clear that each SSO architecture has its strengths and weaknesses, and one should carefully consider the environment before opting for a particular solution. In a closed environment, the focus is likely to be on the deployment, running and maintenance costs, whereas in an open environment the issue of privacy protection may play an equally important role. The integration of SSO, privacy protection services (such as the ones discussed in sections 3.4.2 and 3.4.4) and Identity Management schemes, such as the ones described in [22, 58, 106] remains an active research area. Some of the techniques presented in part II of this thesis can be used to help achieve such an integration.

Chapter 4

An SSO scheme based on GSM/UMTS

Contents

This chapter presents a novel authentication scheme that is built on existing cellular (mobile) telephone technology, in particular the GSM and UMTS systems. Much of the material in this chapter has previously been published in [158].

4.1 Introduction

In 1982 the Conference of European Posts and Telegraphs formed a study group called the 'Groupe Spécial Mobile' (GSM). The purpose of this group was to develop a pan-European system for mobile wireless communications. In 1989 the responsibility for this project was

transferred to the European Telecommunication Standards Institute which published the first specifications in 1990. The first commercial services based on the GSM specifications started in 1991.

Today, GSM stands for 'Global System for Mobile communications'. With over one billion subscribers worldwide, GSM networks exist on every continent of the planet. The reader interested in the history of GSM is referred to [203]. In this chapter we use certain GSM security services in order to build an SSO scheme that works in an open environment of SPs. In our scheme, the GSM network operator acts as the AS, and the user authentication method is similar to that used to authenticate subscribers in a typical GSM network, in that it is based on a secret that is shared between the network operator and the subscriber. The scheme falls into the category of proxy-based true SSO schemes, and requires only minimal changes to the deployed GSM infrastructure. It can potentially be accommodated by the Liberty Alliance specifications as an additional SSO profile.

The rest of this chapter is organised as follows. The next section introduces the necessary GSM terminology and reviews the relevant GSM security services. Section 4.3 describes the proposed protocol for user authentication and SSO using GSM, while section 4.4 analyses the associated security threats. Section 4.5 discusses the advantages and disadvantages of the protocol. In section 4.6 the protocol is extended to cover the third generation of cellular telephony networks, called the Universal Mobile Telecommunications System (UMTS). Finally, sections 4.7 and 4.8 give a summary and an overview of related work.

4.2 The GSM security services

A subscriber to a GSM network is identified and authenticated through the use of a tamper-resistant smartcard, called the Subscriber Identity Module (SIM). The SIM is issued by the subscriber's GSM network operator, and must subsequently be inserted into his Mobile Equipment (ME), typically a mobile telephone handset. The SIM contains a number of data items including the following.

- A cryptographic key, denoted K_i, to be used with a symmetric cryptosystem.
- A key derivation algorithm, denoted $A8$, that takes as input a key K_i and a random

number RAND, and that outputs a session key K_c.

- A unique identifier called the International Mobile Subscriber Identity (IMSI). The IMSI contains a number of fields and can be used to uniquely identify a GSM subscriber on a worldwide basis. Two of the IMSI fields, namely the Mobile Country Code (MCC) and the Mobile Network Code (MNC), uniquely identify the subscriber's GSM operator.

The key K_i, along with the corresponding IMSI, is stored in a server called the 'Authentication Centre' (AuC), belonging to the subscriber's GSM network operator. It is important to note that the keys K_i are never output by either the AuC or their respective SIMs throughout the lifetime of the system. An implementation of the $A8$ algorithm is also available to the AuC. It is interesting to note that this algorithm is operator-specific; different operators may use different $A8$ algorithms and, indeed, the same operator may use different algorithms for different SIMs. GSM operators typically keep the description(s) of the $A8$ algorithm(s) they use secret.

4.2.1 The GSM data confidentiality service

The GSM security service we are interested in, called 'subscriber data confidentiality for the GSM air interface', is used to encrypt the data that is transmitted between the ME and the GSM network. In particular, we are interested in the mechanism that is used to derive the encryption key that is used by this service. As we see below, a certain protocol, which is executed every time the ME attempts to connect to a 'visited' GSM network operator, is used for this purpose. Before describing it we observe that the visited network operator may or may not be the same as the one that issued the SIM to the subscriber. We refer to the SIM issuer as the subscriber's 'home network' operator.

We now describe the protocol according to which a secret session key is generated by the subscriber's SIM and the visited network. (Only the relevant steps are shown; in particular, the process by which the ME is authenticated to the network is omitted. Also omitted is a description of the means by which, in most circumstances, the ME can avoid the need to send the IMSI across the wireless interface. The reader interested in comprehensive descriptions of the security of the GSM air interface is referred to [25, 79, 80, 193, 200]).

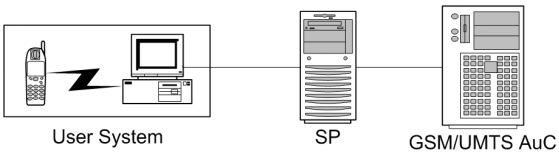

Figure 4.1: PC as network access device.

1. The subscriber's ME extracts the IMSI from the SIM and sends it to the visited network.
2. The visited network looks at the MCC and MNC fields of the IMSI and determines the identity of the subscriber's home network. It sends the IMSI to the home network.
3. The AuC of the home network generates a random number (RAND), finds the secret key K_i and the key derivation algorithm $A8$ that correspond to the received IMSI, and computes the session key as $K_c = \mathrm{A8}(K_i, \mathrm{RAND})$. The values of RAND and K_c are then passed to the visited network.
4. The visited network sends the value RAND to the ME.
5. The ME passes the RAND to the SIM, which computes the session key $\bar{K_c}$ using its copy of K_i and the key derivation algorithm $A8$. The key $\bar{K_c}$ is then output by the SIM to the ME.

Obviously, if the visited network is the same as the subscriber's home network, then step 2 is omitted. After the protocol has completed, the ME uses the key $\bar{K_c}$ to encrypt the data sent to the visited network. The latter attempts to decrypt the received data using the session key K_c. As longs as $K_c = \bar{K_c}$, this process yields correct and meaningful decryptions.

4.3 Using GSM for SSO

This section describes the proposed user authentication method and the associated SSO protocol. The scheme makes use of the infrastructure that supports the GSM session key derivation mechanism described in the previous section. Before describing the SSO protocol, we introduce the entities involved and state the assumptions on which the scheme is based.

4.3.1 System entities

The entities that interact in the SSO scheme are the SP, the GSM operator's AuC, and the User System (US), as described below. We assume that the US is connected, via some form of network, to the SP and hence can exchange messages with it. We do not specify the nature and communication protocols of this network, as our scheme is independent of the underlying technology (although, of course, it will be affected by the network's performance characteristics).

4.3.1.1 Service Provider and Authentication Centre

We assume that the relationship between SPs and GSM operators is regulated by contractual agreements and other non-technical means that are beyond the scope of this scheme. To avoid the need for large numbers of such agreements, the SP/GSM relationship could be established via one of a relatively small number of third parties providing a 'broker' service. Such a third party could be established specifically to support this SSO scheme. It is also assumed that the SPs and the AuC have the means to establish an authenticated and integrity protected communications channel. This is necessary in order to protect against the attacks described in section 4.4.5 below, and might involve routing all communications via the trusted third party that is brokering relationships between SPs and GSM operators.

In the scheme described here, the subscriber's GSM home operator acts as the AS for a number of SPs. As in any true SSO scheme, the AS needs to be trusted for the purposes of authentication by both the end-users and the SPs.

4.3.1.2 User System

The US consists of a network access device, a SIM and a SIM reader. The network access device might be a desktop or laptop computer, with some form of compatible ME, e.g. a GSM telephone handset, as the SIM reader. The computer and ME need to be interconnected, e.g. using a cable, infrared, Bluetooth[4.1] or a Wireless Local Area Network (WLAN) [98]. Regardless of the method used, we assume that this link is protected against eavesdropping. The corresponding configuration is shown in Figure 4.1. Although the figure implies that

[4.1] www.bluetooth.com

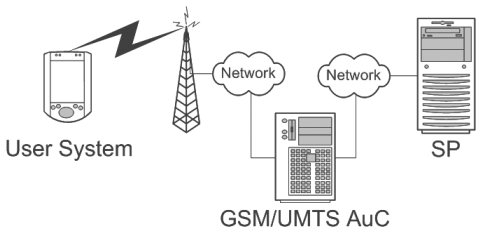

Figure 4.2: Combined mobile equipment and network access device

the US is connected to the network via a wireless link, this does not necessarily have to be the case. Alternatively, the access device and SIM reader could be combined, e.g. in a Wireless Application Protocol[4.2] enabled device — see Figure 4.2.

The US is the entity through which the end user authenticates to the AuC and subsequently achieves SSO at different SPs. It should be noted that no direct communications between the US and the AuC take place in the proposed protocol; messages are 'routed' through the SP. However, the US and the AuC need to agree on and implement a Message Authentication Code (MAC) function (see section 1.1.4.2).

4.3.2 The authentication and SSO protocol

The proposed protocol starts whenever the SP wishes to authenticate the user, e.g. when the user requests a protected resource from the SP; it is also used whenever the SP decides that the user has to be re-authenticated. In order to prevent the attack described in section 4.4.4, we assume that, prior to the execution of the SSO protocol, the user has authenticated the SP and, as a result of this, a cryptographically protected session is set up between the SP and the user machine. This can be achieved, for example, using SSL/TLS with server-side certificates [180]. The protocol consists of a series of four messages that are exchanged as follows:

1. SP → US: RAND
2. US → SP: IMSI, MAC_{K_c}(SPID)
3. SP → AuC: IMSI, MAC_{K_c}(SPID), RAND
4. AuC → SP: Authentication Assertion or Failure Notification

[4.2] www.wapforum.org

In message 1, the SP sends a random challenge (RAND) to the US, where RAND conforms to the requirements expected of a GSM RAND value. That is, it should contain 128 bits and it should be chosen such that the same value is never used twice by a particular SP (except with negligible probability), and that the values chosen can never be predicted (again except with negligible probability). The US forwards this to the SIM, which then computes a secret session key K_c as described in section 4.2 above. The key is returned to the US. The US must also extract the IMSI from the SIM at some point — this can be done with a standard 'call' to the SIM. The IMSI uniquely identifies the home network and the user's subscription.

The US then computes a MAC on the unique identifier (SPID) of the SP that wishes to authenticate the user, using key K_c. It should be noted here that, the user should have authenticated this SPID during the process of SP-to-user authentication. Moreover, as in [117], K_c is used for MAC computation while it was designed for data encryption. If this breach of the key separation principle is a concern, then K_c could be passed though a one way function (see section 1.1.2) or some other appropriate key derivation function before being used for MAC computation. It should also be noted that the MAC scheme should be unforgeable for all possible values of SPID. This means that it should be infeasible, without knowledge of K_c, to construct a valid (SPID, MAC) pair, for any possible value of SPID. This is weaker than the usual security notion for a MAC scheme. The usual notion of 'existential unforgeability' [15] implies the notion we require and, thus, any existentially unforgeable MAC scheme is suitable for use in the protocol. The reason for introducing this MAC in the protocol is to bind the user authentication to the SP authentication. This results in the user and the SP mutually authenticating each other, which is necessary in order to protect against the attack described in section 4.4.4. Note that the authentication scheme of the GSM over-the-air interface does not provide mutual authentication between the user and the SP, and hence using that scheme is likely to be inappropriate in the context of SSO.

The US constructs message 2 which consists of the IMSI and the MAC and sends it to the SP.

The SP examines the Mobile Country Code (MCC) and Mobile Network Code (MNC) fields in the received IMSI to determine the address of the AuC of the user's GSM home network operator. In message 3 the SP simply forwards the IMSI and the MAC it received from the US to the AuC, and appends the RAND from message 1. In terms of the Liberty

Alliance specification [133], message 3 corresponds to an *authentication request*.

The AuC finds the secret key K_i and $A8$ algorithm corresponding to the IMSI of message 3 and derives K_c using the given RAND. It then computes a MAC on the SPID of the SP from which it received message 3, using K_c (or a key derived from K_c). If the resulting value matches the MAC received in message 3, then the user is deemed authenticated at the AuC. In that case, the AuC sends an authentication assertion to the SP in message 4. Otherwise, message 4 is a failure notification. In Liberty Alliance terms this message corresponds to an *authentication response*.

In this scheme, SPs differentiate between users based on their IMSIs. Thus, the protocol can also be used for initial user registration; whenever an SP encounters a 'new' IMSI, it creates a new account for that user. Whether or not individual SPs keep more information about users in their accounts, and how this is mapped to their IMSIs, is outside the scope of the scheme.

In the US, the protocol might be implemented as a continuously running process (also known as a 'service' or 'dæmon') or as part of the client software that is used to access the service offered by the SP. In the SP, the protocol would have to be implemented by the software that offers the service (the 'server software'). A File Transfer Protocol (FTP) SP, for example, would have to support the protocol at the FTP server software level. The GSM operator would most likely implement the protocol as a process that runs continuously on a dedicated server.

4.4 Threat analysis

This section considers threats to the scheme and corresponding countermeasures. Each potential attack is considered separately.

4.4.1 Stolen SIM attack

Assuming that the SIM is not protected by a PIN, an adversary with a stolen SIM is potentially able to make fraudulent phone calls at the legitimate owner's expense, for as

long as the SIM is not reported stolen and blocked. Typically, SIMs are stolen together with the ME, and the thief also thus gains access to any personal information that is stored in the ME. In the context of the scheme described above, the thief can also impersonate the user to all SPs that support SSO using the proposed protocol. For low or medium value services (such as online forums and personal e-mail) this risk might be acceptable, compared to the costs associated with SIM/ME theft. However, for high value services (e.g. online banking, e-commerce or business email) the scheme may need to be combined with another authentication method, e.g. username and password. Such a combination would result in two-factor authentication; the attacker would need both the SIM and the user's password in order to impersonate the user to an SP. Also, once the user realises that the SIM has been stolen (typically soon after the incident) and reports this to the GSM operator, the subscription will be blocked and impersonation prevented.

4.4.2 SIM cloning attack

A SIM can be cloned or emulated through software if the secret key K_i can be extracted from it. An attack exploiting a weakness in the COMP128v.1 algorithm, which was used as the $A8$ key derivation function by some GSM operators in early SIMs, enabled such key extraction [203]. Cloned SIMs enable the adversary to impersonate subscribers at SPs (including the GSM network) in much the same way as stolen ones. Moreover, the victim of a cloned SIM is less likely to detect the attack than the victim of a stolen SIM. We therefore have to assume that the $A8$ key derivation function is sufficiently secure to prevent such an attack, and that appropriate measures have been put into place by the SIM manufacturer to prevent an adversary from extracting the secret key K_i from it.

4.4.3 Compromise of privacy

Unlike the protocol described in [117], the scheme described here does not involve the user's Mobile Subscriber Integrated Service Digital Network number (i.e. his telephone number) or any personally identifying information other than the IMSI. As the mapping between the IMSI and other user data is kept at the trusted AuC, an SP only learns the identity of the user's home GSM operator; this information is necessary anyway because the SP needs to be able to contact the user's GSM home network — see section 4.3.2. Given that each GSM operator has many subscribers, learning a user's operator is not likely to enable

an SP to discover the real identity of a user, as this identity will remain hidden within a large anonymity set [164].

If dishonest SPs collude they can unambiguously correlate information about common users without their consent, using their IMSIs. As is the case generally with true SSO schemes, the scheme does not address this issue. Even in the Liberty specifications, where different, opaque user handles must be used for each AS/SP association [130], dishonest SPs can still link identifiers with the help of the AS. It appears that, in order to prevent identifiers being linkable, special privacy-preserving cryptographic primitives need to be used that have not been widely used in commercial systems. Part II of this thesis introduces a scheme that uses such primitives in order to provide this unlinkability.

4.4.4 Forwarding attack

Without SP-to user authentication prior to execution of the SSO protocol, the scheme is subject to a special type of 'man-in-the-middle' attack, which we call a 'forwarding' attack, as follows. An adversary could forward message 1 received from an SP as part of the authentication process to a 'victim' user, while masquerading as the SP to the user (for example by spoofing the SP's network address). Forwarding the user's valid response (message 2) to the SP might enable the adversary to impersonate the victim user at that SP.

Without SP-to-user authentication occurring prior to executing the SSO protocol, the protocol provides only user-to-SP, rather than mutual authentication. In this setting the above attack therefore applies and is of practical relevance and a real threat in many situations; it is therefore highly desirable for the user, before releasing message 2, to authenticate his communication partner as well. In other words, it is necessary, in these situations, for SP-to-user authentication to occur prior to the execution of the SSO protocol described in this chapter. This is why we assume that such authentication precedes every execution of the protocol. As a side comment, note that, as discussed in section 2.1.2, it is likely that a protected communications channel will be required anyway, since the user is requesting access to a protected resource.

4.4.5 Attacks on the SP/AuC Link

An active adversary that modifies network traffic between an SP and the AuC is able to defeat the system, since the AuC will not be able to determine which SP requested an assertion (the origin of message 3 could be altered) and, more importantly, the SP would not be able to have confidence in the authentication assertion (message 4). Origin authentication and integrity protection for messages 3 and 4 is therefore a fundamental requirement. We therefore assume that this network link is appropriately protected, for example by well-established techniques such as SSL/TLS channels with both server and client side certificates [180], IPsec tunnelling, or some other appropriate 'virtual private network' [193]. A permanent secure channel is also potentially beneficial from an efficiency viewpoint, because it can be reused for multiple protocol executions.

4.4.6 Replay attack

An adversary could capture message 2 of a previous protocol run between the US and an SP. The adversary might later replay that message to try to impersonate the user to the SP. The attack will only succeed if RAND sent by the SP (in message 1) is the same as a previous one. It is therefore important for SPs to use numbers with good randomness properties, such that the probability of challenging a given user with the same number twice is negligible. An SP that does not follow this policy, however, only renders itself (and not other SPs) vulnerable to this kind of attack.

4.4.7 Attacks against the authentication centre

As for any AS in an SSO scheme, the AuC constitutes a central point of failure, and is therefore a component highly susceptible to service denial attacks. The AuC, at the same time, is the only entity trusted by both end-users and SPs. It is assumed that the GSM network operator will not abuse this trust and that the AuC will be well-protected against service denial and illegal access. This latter assumption is likely to hold in any event, as a failure of the AuC would also bring down the entire GSM network.

4.5 Advantages and disadvantages

This section briefly discusses the advantages and disadvantages of the proposed authentication method and SSO protocol.

Advantages resulting from the synergy of combining GSM and SSO include the following.

- The scheme does not require user interaction. This yields transparent user authentication at the SP. One envisioned scenario is that a user approaches a (public) network access device and is transparently logged into one or more SPs without having to take his mobile phone out of his pocket.
- As the authentication method is transparent, it can be repeated whenever appropriate. For example, an online banking SP could, at any time during a session, request user re-authentication for every sensitive resource requested. This increases the level of security achieved without usability implications.
- As the user's SIM is used for every (re)authentication, there is a simple single logout mechanism. Once the user leaves, re-authentication will fail and the SP can log the user out without manual interaction.
- No sensitive information (such as usernames, passwords or cryptographic keys) is sent over the network. This protects the user's privacy. Furthermore, no risks of information exposure arise at the SP.
- The protocol itself does not impose major computational overheads on any of the involved parties, although there may be a cost associated with the countermeasures against the attacks described in section 4.4.
- Changes in the deployed GSM infrastructure are minimal. In particular, the SIM does not need to be modified in any way.
- The GSM operator could provide the SPs with user profile information (with the user's consent). This would enable SPs to offer location based services, or automatic form completion for e-commerce transactions.
- As GSM operator fraud management systems use the IMSI to refer to 'suspect' subscriptions, fraud detection can easily be extended to cover SPs.

The scheme comes with the following disadvantages.

- The scheme currently has no formal analysis or 'proof of security', e.g. a complexity theoretic reduction to some hard computational problem. This is because, for the standard adversarial model [17], the countermeasures discussed in sections 4.4.4 and 4.4.5 need to be present. If these were to be treated as part of the scheme, the resulting complexity would make the scheme difficult to analyse formally. Furthermore, the key derivation function $A8$ resides on the SIM (and therefore cannot be replaced) and potentially differs from one GSM operator to another. A formal analysis would have to either consider multiple such functions or resort to an abstraction that might not be very realistic.
- A GSM operator can only provide authentication for its subscribers. The system therefore works only for subscribers of GSM network operators and those SPs that have contractual agreements with those operators.
- SPs may be charged by GSM operators for the user authentication service. While this is not a disadvantage for the operators (indeed it may provide a useful additional revenue stream for operators), SPs must weigh the cost against the benefits gained from the proposed scheme.

4.6 Using UMTS/3GPP for SSO

A variant of the protocol is now described that is based on the Universal Mobile Telecommunications System (UMTS) of the Third Generation Partnership Project[4.3]. Like GSM, UMTS authentication is based on a secret key (K_i) shared between the subscriber and its home network operator. The main difference is that the network sends an 'authentication token' (AUTN) as well as the RAND to the subscriber's ME [25]. As this AUTN can only be produced by the user's home network AuC, in UMTS the network also authenticates itself to the user. The proposed variant protocol operates as follows.

1. US → SP: IMSI
2. SP → AuC: IMSI

[4.3] `www.3gpp.org`

3. AuC → SP: RAND, AUTN
4. SP → US: RAND, AUTN
5. US → SP: MAC_{IK}(SPID)
6. SP → AuC: MAC_{IK}(SPID), IMSI, RAND
7. AuC → SP: Authentication Assertion or Failure Notification

The US sends its IMSI to the SP (message 1) which forwards it to the AuC (message 2). The AuC generates a random challenge (RAND) and computes the AUTN as a function of RAND, the secret key K_i that corresponds to the given IMSI, and a counter value specific to the SIM. The RAND and AUTN values are sent back to the SP (message 3) and forwarded to the US (message 4).

The US's SIM checks the validity of AUTN and, if valid, generates an 'integrity key' IK as a function of RAND and its secret key K_i, just as it does during normal UMTS authentication [25]. The US uses this key (rather than the encryption key K_c) to compute a MAC on the SP's unique identifier (SPID).

The rest of the protocol is similar to the GSM version. In message 5 the US sends the MAC to the SP which forwards it to the AuC in message 6, while appending the US's IMSI and the RAND of message 3 (message 6 corresponds to a Liberty *authentication request* [133]). The AuC derives the integrity key IK from the RAND and the secret K_i corresponding to the given IMSI. Using the IK, the AuC computes a MAC on the SPID of the SP from which it received message 6. If it matches the MAC of message 6 then the authentication process is deemed successful. If this is the case, then the last message, which corresponds to a Liberty authentication response, contains an assertion; otherwise it contains a failure notification.

As in the GSM version of the protocol, the US must authenticate the SP prior to use of the protocol, and communications between SP and AuC must be mutually authenticated and their integrity protected (see sections 4.4.4 and 4.4.5).

Some of the 'side effects' when using the protocol described in this section, as compared to its GSM version described in section 4.3.2, are the following:

- As the RAND is generated by the AuC instead of the SP, the risk of poor random number generation (see section 4.4.6) is taken away from individual SPs. Associated computational costs, however, are centralised at the AuC.
- The number of messages has almost doubled. This could lead to increased response times.
- Validation of the AUTN (message 4) by the US provides for AuC authentication, integrity and freshness assurance [25]. As a result, previous (RAND, AUTN) pairs cannot be reused. The process is thus in this case no longer independent of subscriber authentication over the UMTS air interface.

4.7 Related work

Claessens et al. [57] propose a GSM-based authentication method for the World Wide Web. The proposed scheme, however, makes extensive use of another GSM service that requires manual interaction from the user, called the Short Messaging Service (SMS). Several commercial SSO solutions, including Ubisecure's[4.4] Ubilogin™ and Entrust's[4.5] Truepass™, also involve the SMS.

The e-commerce user authentication protocol proposed in [117] relies on the fact that GSM subscribers and network operators share a secret key K_i. The same fact is exploited by the protocol proposed in this chapter. Being an e-commerce protocol, however, the scheme in [117] discloses information to the merchant (i.e. the SP) that does not need to be disclosed in an SSO scenario. The same applies to the 'Murabaha transaction' scheme described in [4], which also makes use of the SMS.

In an independently developed Internet draft [102], the use of the GSM security mechanisms described in section 4.2 for user authentication to arbitrary networks services is specified. The protocol proposed in the draft is similar to the one proposed in this chapter, in that it makes use of a MAC computed using a key derived from the secret key K_i. However, it requires six message exchanges between the US and the SP while the protocol proposed in this chapter requires only two. Nevertheless, the protocol in [102] also establishes a session key between the two endpoints, has a number of options (e.g. the use of a temporary pseudonym

[4.4] www.ubisecure.com
[4.5] www.entrust.com

rather than the IMSI and a faster re-authentication variant) and also encapsulates the messages within the Extensible Authentication Protocol (EAP) [2]. In [152] (an earlier version of) the protocol in [102] is combined with other protocols in order to facilitate GSM-based user authentication in access networks.

4.8 Summary

We have proposed user authentication schemes for distributed systems where a GSM or UMTS operator provides authentication assertions to relying SPs. The schemes can potentially be specified as Liberty Alliance profiles, thereby conforming to this open specification.

The protocol yields a seamless user experience as no user interactions are needed, allowing several transparent re-authentications to occur within a user/SP session. This leads to an equally seamless, secure and simple single logout mechanism. The protocol preserves user privacy and mobility. To protect against the risks associated with SIM theft or cloning, the scheme can be complemented by an additional mechanism such as username and password. Impersonation will then only succeed if the attacker has access to both the user's SIM and password: the result would be two-factor authentication.

The required changes to the existing GSM or UMTS infrastructure are minimal. The scheme uses existing SIMs and appropriately equipped MEs. The GSM or UMTS operator's AuC, the SPs and the US need only support the protocol at a software level. Computational overheads are small.

Chapter 5

An authentication scheme based on credit/debit smart cards

Contents

This chapter presents a novel interdomain user authentication scheme that is based on payment (credit or debit) smart cards that conform to the Europay MasterCard Visa (EMV) specifications. The card itself, in conjunction with the user's access device and the EMV infrastructure, plays the role of the AS. The associated SSO protocol does not require online card issuer participation, preserves user mobility, and does not put the user's financial data at risk. Much of the material in this chapter has previously been published in [155].

5.1 Introduction

Smart cards can be used in the context of electronic commerce to provide security and mobility [195]. In a global marketplace, systems need to be compatible and must limit the opportunity for fraud. This is why, in 1994, three companies, namely Europay International, MasterCard International[5.1] and Visa[5.2] International, formed a joint working group in order to sponsor the production of a global specification for smart card-based electronic financial transactions. The first set of specifications was released in 1998. In the following year, a new organisation, called EMVCo[5.3], was formed by the same players in order to manage, maintain and enhance the specifications, to ensure interoperability and acceptance of the associated smart cards on a worldwide basis, and to oversee a 'type approval process' that defines test requirements and test cases that are used to test the associated card terminals for compliance. Today the members of the EMVCo organisation are JCB Co.[5.4], Ltd., MasterCard International and Visa International. Each of these members owns one third of EMVCo.

In December 2000, EMVCo released a set of Integrated Circuit Card (ICC) specifications for Payment Systems [75, 76, 77, 78], essentially an update of the previous specifications. The focus of these specifications is on the card/terminal interactions that take place at the Point of Sale during a financial transaction.

This chapter presents a scheme in which EMV-compliant cards provide user (i.e. cardholder) authentication, and proposes an associated protocol that achieves SSO at SPs that may lie in different administrative domains. In the scheme, the card itself, in conjunction with the cardholder's access device, acts as the AS. According to the taxonomy of chapter 3, the scheme is a local true SSO scheme. It can be regarded as an alternative to other smart card-based authentication schemes, for example schemes that rely on SIM cards (such as the scheme of Chapter 4).

The remainder of this chapter is organised as follows. The next section contains a review of relevant EMV architectural components and security services. Section 5.3 describes the proposed scheme and protocol, while section 5.4 analyses the associated security threats.

[5.1] http://www.mastercard.com
[5.2] http://www.visa.com
[5.3] http://www.emvco.org
[5.4] http://www.jcbinternational.com

Section 5.5 discusses advantages and disadvantages, and sections 5.6 and 5.7 give an overview of related work and conclude this chapter.

5.2 Review of EMV security services

This section introduces the relevant components of the EMV specification. For a full description the reader is referred to [75, 76, 77, 78, 174].

There are four major interacting entities in the EMV payment system, namely the cardholder, the merchant, an acquiring bank (the 'Acquirer'), and the card issuing bank (the 'Issuer'). The specifications focus on the interactions between card and merchant terminal. When the card is inserted into the terminal, the steps that occur include the following.

1. The terminal selects the appropriate EMV application by issuing a SELECT command [75, p.65] to the card.
2. The terminal initiates 'Application Processing' by issuing a GET PROCESSING OPTIONS [77, p.19] and a number of READ RECORD [77, p.23] commands. The purpose of this step is to enable the card and the terminal to exchange the necessary data for the rest of the transaction.
3. The terminal performs 'Processing Restrictions' [77, p.48]. This mandatory step does not involve communication with the card — its sole purpose is to provide assurance of application compatibility between terminal and card.
4. The terminal issues an INTERNAL AUTHENTICATE [77, p.21] command to the card. This optional step initiates 'Offline Data Authentication', which can be either Static Data Authentication [76, p.15] or Dynamic Data Authentication [76, p.24]. The purpose of this step is to verify the card's authenticity.
5. The terminal performs 'Cardholder Verification' [77, p.50]. During this optional step the cardholder's Personal Identification Number (PIN) is verified, either offline to the card (using the VERIFY command [77, p.25]), or online by the Issuer.

We next focus on the Dynamic Data Authentication (step 4) and the Cardholder Verification (step 5) processes, defined in the EMV specifications. These steps are of particular interest

since, in the scheme proposed below, the card reader is under the control of the cardholder, rather than the merchant terminal, as would be the case at the point of sale.

5.2.1 Dynamic Data Authentication (DDA)

DDA is supported by a PKI (see section 1.1.5), as specified in [76]. In particular, every DDA-capable card has its own asymmetric key pair which is generated and loaded on the card by the Issuer. While the private key cannot be extracted from the card, its public counterpart can be retrieved using the READ RECORD command. This public key is embedded in a public key certificate which is signed by the Issuer. A certificate for the Issuer public key needed to verify the card-specific public key certificate, signed by the Payment System's top-level Certification Authority (CA), is also stored in the card and can be retrieved by the terminal. As a result, the merchant terminal only needs to maintain an accurate copy of the public key of the Payment System CA in order to verify the Issuer's, and hence the card's, public key certificates, and finally any data signed by the card itself. CA public key management principles and policies for the EMV system are defined in [76].

A simplified description of Dynamic Data Authentication (DDA) is as follows:

1. The terminal retrieves the Issuer and card public key certificates from the card. The Issuer certificate is verified using the appropriate trusted root public key of the Payment System's top level CA. The card certificate is then verified using the public key obtained from the Issuer certificate.
2. The terminal issues an INTERNAL AUTHENTICATE command to the card. The command requires a number of parameters, including a random, terminal-generated nonce (see section 1.1.1).
3. The card computes a digital signature over the terminal-provided data (including the nonce) and 'card Dynamic Data', i.e. data generated by and/or stored in the card [76, p.35]. The card outputs both the signature and the card Dynamic Data.
4. The terminal reconstructs the signed data structure and verifies the signature using the card public key retrieved and verified in step 1.

The DDA mechanism guarantees data integrity and freshness to the terminal (i.e. it provides

assurance to the terminal that the 'card Dynamic Data' has not been tampered with and that it corresponds to the current communication session). Assuming tamper resistance of the card and the soundness of the Issuer's security procedures, DDA also provides card authentication and card counterfeiting prevention. It should be noted that not all EMV-compatible cards are DDA-capable.

5.2.2 Cardholder verification

The identity of the cardholder is verified using a PIN. The PIN is entered into the terminal, and may then either be verified online by the Issuer or offline to the card. In the latter case the terminal issues a VERIFY command to the card which takes the PIN as a parameter. The PIN, when submitted to the card, may or may not be encrypted. If encryption is required, then it is performed using the card public key. The card checks whether the supplied PIN matches the one stored inside the card, and responds accordingly. If the number of unsuccessful offline PIN verification attempts exceeds a certain limit, the PIN on the card is blocked and can only be unblocked using a script sent to the card by the Issuer, where the security of the script is protected using a secret key shared by the card and the Issuer.

5.3 Using EMV cards for SSO

This section describes the proposed EMV-based SSO scheme.

5.3.1 System entities

The entities involved in the authentication/SSO scheme are the Cardholder System (CS), the card itself, and the SPs.

As briefly mentioned above, the system requires the cardholder system and the card to collectively act as the AS. Instead of the cardholder being directly authenticated by every SP, the cardholder is authenticated by the card, and the card then vouches for the identity of the cardholder to every SP. The fact that the CS would typically consist of a 'standard' PC,

PDA or mobile phone equipped only with a special SSO application and an EMV card, means that this offers an inherently mobile SSO solution; the SSO application could be downloaded from a trusted source when required, and the EMV card is inherently mobile.

5.3.1.1 The cardholder system

The CS consists of the user's (i.e. the cardholder's) network access device and a card reader. A typical configuration would be a PC with an integrated card reader. Whether or not the card reader is equipped with its own (trusted) keypad is optional (see section 5.4.4). Alternatively, the CS could be a wireless network access device (such as a Personal Digital Assistant (PDA) or a 3GPP[5.5] mobile phone) capable of communicating with EMV cards. The CS also needs certain special software that implements the SSO protocol described in section 5.3.4 below. This 'SSO agent' might be realised as a process that continually runs on the CS (also known as 'service' or 'dæmon'), or as part of the software that is used to access the SP (e.g. the web browser, instant messenger, e-mail client, etc.). In this latter context the SSO agent could be uploaded to the CS as an applet running within the SP access software, e.g. as a Java applet running within the web browser, or a Java MIDlet that is delivered over-the-air to a mobile phone. The SSO agent is likely to be provided by an EMV card issuer or a trusted third party.

5.3.1.2 The card

The proposed EMV-based user authentication scheme imposes certain requirements on the cards, as follows. Cards must be DDA-capable. Unfortunately, from the SSO perspective, the public key certificate used during DDA binds the card's public key to the cardholder's Primary Account Number [76, p.33]. It would constitute a potentially serious security and privacy threat if the Primary Account Number was to be used in an open environment such as the Internet. Therefore, the cards used in the scheme described here need to possess a separate, dedicated, EMV application, which we call the *card authentication application* (AA). In the AA, the Primary Account Number (and any other personally identifying information) must be replaced with an identifier (pseudonym) that is not linked to the cardholder's credit/debit account (and cannot be used for financial transactions).

[5.5] http://www.3gpp.org

This implies that the Issuer has to provide the AA with an additional certificate for the public key belonging to the card, although we assume that the same key pair is used (i.e. the card only needs to use a single asymmetric key pair). In the scheme proposed here the certificate serial number is used as the unique cardholder identifier, i.e. no separate user identifier is needed in the certificate created for the AA (and the PAN field can be left blank). Indeed, depending on the application environment, it may be desirable to ensure that the certificate created for the AA (which we call the AA certificate) does not contain any personally identifying information for the cardholder. As the AA certificate serial number can be used for user identification at SPs, it does not need to contain any other information about the user (e.g. a name), thereby giving a measure of anonymity to users of this SSO scheme. The PIN used by the AA should be separate from the PIN(s) used by EMV applications that may coexist on the card.

A further assumption regarding the card is that the AA should be able to maintain state within the current session. In particular, it should be able to maintain a data element that indicates whether or not offline PIN verification (via the VERIFY command) has been performed during the current session, and, if so, the data element should also indicate whether or not PIN verification was successful. This card-provided *PIN Verification Data Element* (PVDE) shall be included in the data that is signed by the card during DDA, as part of the card Dynamic Data.

It should be noted here that a card session begins with Application Selection (step 1 in section 5.2) and ends when the card reading device deactivates the card [75, p.17]. This latter event includes premature removal of the card from the reader.

5.3.1.3 Service providers

In the proposed SSO scheme, SPs are required to obtain and store the public keys of the CAs of the EMV Payment Systems that are to be supported (and trusted). This requirement is exactly the same as that applying to merchant terminals for 'standard' use of EMV cards. The management, distribution and revocation of these public keys is outside the scope of the scheme, but the principles are similar to those specified in [76] for merchant terminals.

It is further assumed that SPs require a user to be authenticated before granting access to protected resources. Instead of executing an authentication protocol directly with the

user, SPs acquire the necessary authentication assertions from the CS, according to the protocol described in section 5.3.4 below. Moreover, as users also need to authenticate SPs, it is necessary that every SP possesses a *unique, human-readable identifier* (which we call SPID).

5.3.2 Trust relationships

The SSO scheme depends on the EMV cards offering a level of tamper-resistance, since these cards act as a trusted computing module within the CS. In addition, cardholders need to trust that SPs will not collude in order to compromise their privacy (see section 5.4.1). Cardholders and SPs also need to trust that

- the Payment System's CA(s) will not impersonate cardholders,
- card Issuers will not impersonate cardholders.

From the cardholder perspective, authentication/SSO can be achieved with those SPs that choose to trust the Payment System CA corresponding to the cardholder's card. From the SP perspective, authentication/SSO can be facilitated only for those cardholders whose Payment System top level CA that particular SP has chosen to trust. The architecture does not provide for explicit trust management at the Issuer level. This feature is inherited from the EMV PKI, which does not allow merchant terminals *not* to trust individual Issuers that have been certified by a trusted CA. This arises from the fact that EMV was designed for use within a closed environment in which all parties (Issuers and Acquirers) have signed an agreement with the brand. Indeed, even in the non-electronic world, merchants are typically required to accept all cards bearing the brand as part of the condition of being an approved merchant.

5.3.3 The registration protocol

This section describes the registration protocol associated with the scheme. It is initiated by the user. Its purpose is to create a new user/SP association. As explained above, the user's pseudonym with the SP is the AA certificate serial number. So, in principle, it would be sufficient if this identifier is sent to the SP. This would involve the following steps.

1. The CS selects the AA on the card, initiates application processing and performs processing restrictions, as explained in steps 1-3 in section 5.2.
2. Using the READ RECORD command [77, p.23], the CS reads the AA certificate from the card, selects its serial number and sends it to the SP.
3. The SP checks whether there already exists an association with the received serial number and, if not, it creates a new user account for that number.

The above protocol does not prevent a simplistic service denial attack where an adversary registers bogus identifiers in order to abuse SP resources or to cause confusion by registering the identifiers of legitimate users. For this reason it may be desirable to execute a full SSO protocol (described below) for registration. In this case, registration is treated like any other service that requires user authentication.

5.3.4 The SSO protocol

This section describes the SSO protocol associated with the scheme. It starts when the user requests a protected resource from the SP, or for initial registration, in the case where user authentication is required. In order to prevent the attack described in section 5.4.2 below, we assume that the user has authenticated the SP at this point and that, as a result of this, a cryptographically protected session has been set up between the SP and the user machine. This may be achieved, for example, with an SSL/TLS channel with server-side certificates [180].

A detailed description of the protocol follows.

1. The SP sends an authentication request message to the CS. This message contains a freshly generated random nonce and an indication saying whether or not PIN verification is required.
2. The CS selects the card AA, initiates application processing and performs processing restrictions, as explained in steps 1-3 in section 5.2. If this step fails, SSO also fails.
3. If PIN verification was required in step 1, the CS performs offline PIN verification with the card, as explained in section 5.2.2.

4. The CS performs DDA with the card. The main difference from the 'standard' DDA (as explained in section 5.2.1) is that the nonce used with the INTERNAL AUTHENTICATE command is the SP-provided nonce from step 1. The SP's Identifier (acquired during SP-user authentication, as explained above) is also included in the data passed to the card for signing. It should be noted that the CS cannot verify the card's and the Issuer public key certificates as it does not have the Payment System CA's public key.

5. The CS sends an authentication assertion message back to the SP. This message includes the following data structures obtained during step 4.

   - The AA certificate obtained from the card.
   - The Issuer public key certificate.
   - The signature produced by the card as part of DDA. This signature is computed as a function of the nonce sent in step 1, the SPID and the PVDE, as explained in section 5.3.1.2.
   - Any other data that is input to the card signature calculation (including the nonce).

6. The SP verifies the Issuer and card public key certificates, as explained in section 5.2.1. The SP also makes sure that the card has not been blacklisted and that the aforementioned certificates have not been revoked. If this step fails, SSO also fails.

7. The SP reconstructs the data structure that was signed by the card in step 5 and verifies the signature using the card's public key. If verification is unsuccessful, SSO fails.

8. The SP checks the data used to compute the card's signature. In particular, the SP checks the SPID and makes sure that it indeed represents this SP (and not any other). Furthermore, the SP checks the PVDE to ensure that, if required, PIN verification has successfully completed. If the SP's requirements are met, SSO succeeds and access to the protected resource is granted. Otherwise, SSO fails.

The response from the CS (step 5) does not contain any personally identifying information about the cardholder. As discussed in section 5.3.4, the SP can, however, differentiate between users by combining the Issuer-unique serial number, included in the AA certificate for the card, with the Issuer identifier associated with the Issuer certificate.

The CS can achieve SSO at disparate SPs by running the protocol whenever needed. Of course, the card needs to be in the card reader of the CS during the protocol run. If PIN verification has been performed, the card needs to remain in the reader between protocol runs so that the session state is maintained within the card.

5.4 Threat Analysis

In this section possible threats to the scheme are evaluated.

5.4.1 SP collusion

If a number of SPs collude, they can trivially compromise user privacy by correlating the unique identifying (Serial Number, Issuer Identifier) pairs found in the AA certificates. The scheme does not address this threat.

However, as also pointed out in [131], complete prevention of a 'SP collusion' attack is difficult as SPs may also be able to correlate users based on other profile information they may maintain (such as names or telephone numbers). As stated in the Liberty specifications [131, p.71], 'The only protection is for Principals [users] to be cautious when they choose service providers and understand their privacy policies'.

5.4.2 Man in the middle attack

In a fashion similar to the attack discussed in section 4.4.4, an attacker could forward the authentication request message (step 1 of the protocol) received from an SP as part of the SSO process to a victim user, while masquerading as the SP to that user (e.g. by spoofing the SP's interface and SPID). Forwarding the user's valid response (step 5) to the SP might result in successful impersonation.

This attack is prevented by our assumption (see section 5.3.4) that the SP has been authenticated to the user (cardholder) prior to the execution of the SSO protocol. The mechanism on which this authentication is based is outside the scope of the scheme but, as discussed

in section 2.1.3, it should require the user to manually verify the SPID. Ideally, a cryptographically protected session should be set up between the SP and the CS as a result of that authentication. A suitable mechanism for SP authentication is, for example, an SSL/TLS channel with server-side certificates. In fact, since the user requests a protected resource, it is likely that an SSL/TLS connection will be required anyway [180]. In this case the SP's unique identifier, the SPID, would be a field in its SSL/TLS certificate (typically its unique DNS name). Here it is worth noting that, if SSL/TLS is used for SP-to-user authentication, the scheme essentially uses the somewhat ad hoc PKI established on the Internet for SSL/TLS connections to facilitate SP-to-user authentication, and the EMV PKI established for credit/debit card payments to facilitate user-to-SP authentication. In other words, this setting effectively integrates EMV and SSL/TLS certificates into a single mutual authentication scheme.

5.4.3 Traffic analysis

An attacker capable of monitoring network traffic between the CS and SPs could compromise the user's privacy in that the attacker will learn which SPs the user is communicating with. The attack cannot be prevented by encrypting traffic, as packet headers typically need to be available in unencrypted form for routing purposes. This threat is outside the scope of the scheme described here, but could be addressed separately using anonymising techniques, such as those described by Chaum [45].

5.4.4 Attacks using a malicious Cardholder System

The EMV specifications make no provision for cards to authenticate merchant terminals prior to releasing information. As a result, when the card is inserted into the CS, it may be possible for malicious software in the CS to extract private information (such as the cardholder's Primary Account Number, which is likely to be stored in the card) and to disclose it to unintended parties. Similarly, if the card reader does not have its own (trusted) PIN pad, the CS could collect the cardholder PIN used in conjunction with the AA and disclose it to unintended parties.

Clearly, the scope of the attacks that may be launched by a malicious CS extend beyond the scope of the SSO scheme described here. For example, a malicious CS could spoof the

local interface, for example by asking the user to input his PIN at times when this is not necessary (and subsequently disclose it to unintended parties), or spoof remote interfaces (e.g. the look of a SP's web page) in order to trick the user into taking actions that he would otherwise not take. Furthermore, a maliciously modified CS can abuse the user's authentication status at SPs by modifying traffic or hijacking an entire session, even if communications are 'cryptographically protected'.

Thus, the SSO agent has to be trusted by the user not to engage in malicious behaviour, and its integrity thus needs to be protected. Having it digitally signed by a party trusted by the user (e.g. the card Issuer) might address the threat, but risks remain if other malicious software is executed on the CS.

However, despite these threats, the scheme provides two-factor user authentication (proof-of-possession of the card and, if required, proof-of-knowledge of the PIN), even in the presence of a malicious CS. Firstly, it requires the EMV card to be present in the CS; without it user authentication (and therefore illegitimate impersonation) will fail. Secondly, the scheme protects against the CS falsely pretending that PIN verification took place; the PIN verification status maintenance is managed by the trusted card itself, as explained in section 5.3.1.2. This protects against PIN compromise by a malicious CS, as the PIN never needs to be inserted into the device (for SPs that do not require PIN verification).

5.4.5 Stolen EMV card

The theft of an EMV card will allow an attacker to impersonate the legitimate cardholder to SPs that do not require PIN verification. The obvious countermeasure is for SPs to require PIN verification. In this case the attacker will not be able to impersonate the legitimate cardholder, even by using a maliciously modified CS. Of course, if the attacker also has access to the user's PIN, then impersonation will be successful.

In order to guard against 'stolen card' attacks, SPs should follow the same procedures as merchant terminals. In particular they should periodically contact card Issuers and/or Payment System CAs to obtain Certificate Revocation Lists (CRLs) and/or card blacklists. Step 6 in of the SSO protocol (section 5.3.4) provides for checking of these CRLs and blacklisted card information.

5.4.6 Service denial attacks

The scheme requires the SPs to check whether the signature returned by the CS is computed using the correct nonce, i.e. it requires the SP to maintain state while waiting for the response from the CS. This potentially opens the door to service denial attacks [6]. However, we have assumed prior SP-to-user authentication, which ideally results in a secure session. This means that, before the SSO protocol is executed, the SP has established that it is talking to an existing client for whom it has already created some state. The fact that the SP is required to remember a nonce for each user-to-SP authentication attempt, is thus not likely to significantly increase the SP's exposure to service denial attacks.

5.4.7 Signature oracle attacks

The SSO scheme, as described in sections 5.3.1 and 5.3.4 (step 5), involves the card signing a data string containing a nonce supplied by the SP. Thus the protocol involves the card signing a message, part of which is provided by an external party. There exists the possibility that this could be used as part of an 'oracle attack', where the card is persuaded to sign a string that could be used in another application using the same key pair. However, this is not a significant threat in the case of the EMV payment application since the signed string is different in format from the one expected by an EMV application.

5.5 Advantages and Disadvantages

This section discusses the advantages and disadvantages of the described scheme.

5.5.1 Advantages

Advantages of the user authentication scheme described in this chapter include the following.

- The scheme reuses the existing EMV PKI, which is already established on a world wide basis. Whilst the scheme does require one additional cardholder certificate to be

generated and installed on the card, no extra key pairs are required, and the existing Issuer certificates and Payment System public keys are re-used.

- The scheme does not require the continuous online presence of the card Issuer.
- Once the authentication/SSO protocol has completed successfully, subsequent protocol runs do not necessarily require user intervention. This yields transparent user authentication at SPs subsequently used by the cardholder.
- As user authentication is potentially transparent to the cardholder, the protocol can be repeated whenever appropriate. An online banking SP, for example, may wish to ensure that the cardholder's card is still present in the CS whenever access to a sensitive resource is requested. Rerunning the SSO protocol during a session increases the achieved level of security without usability implications.
- No personally identifying information about the user need be included in the messages exchanged. This enables the user's anonymity and privacy to be protected. Furthermore, no risks of personal information exposure arise at the SP.
- Maliciously acting devices can only compromise the user's current session or impersonate users while the EMV card is present. Furthermore, they cannot falsely pretend that PIN verification took place successfully.
- The scheme does not necessarily require an online third party. SPs need, however, to follow the same principles and policies as merchant terminals with respect to the certificates used.
- The scheme preserves user mobility.
- The scheme can potentially be adapted as a new Liberty Alliance profile (see section 3.5.2). This would involve specifying message formats and protocol logic in a way that is compatible with the Liberty specifications.

5.5.2 Disadvantages

Disadvantages of the authentication/SSO scheme described in this chapter include the following.

- Issuers must install a separate EMV application on the card in order to support user authentication. This has a potentially significant cost. This cost is minimised if the AA is installed on the card at the time of issue, and it has to be weighed against the potential benefits gained by the Issuer. These might include new revenue streams from SPs that benefit from use of the AA.
- The cards used must be DDA-capable. The cost of DDA-capable cards is higher than the cost of cards not capable of DDA.
- The user's identifier is the same at all SPs. It is therefore trivial for SPs to combine their knowledge about users.
- The scheme obviously only works for EMV cardholders equipped with card readers. The cost of the card reader (and maintaining the SSO agent), has to be weighed against the convenience offered by SSO.

5.6 Related work

Single sign-on architectures within enterprise environments are examined in [65]. Currently deployed or proposed SSO schemes for open environments are proxy-based, i.e. they require a continually online AS [130, 144, 149]. The scheme proposed in this chapter, on the other hand, does not necessarily require the continuous online presence of any party; according to the taxonomy of chapter 3 it falls into the category of local true SSO schemes. Because it is based on an existing infrastructure and a particular trust model, the scheme also constitutes an interesting alternative to both the aforementioned proxy-based schemes, and the local true SSO scheme described in chapter 6.

Other related work includes [117], where a security-enhanced e-commerce transaction scheme based on EMV cards is proposed. The scheme makes use of DDA and offline PIN verification in order to facilitate card and cardholder authentication respectively. In contrast to the scheme proposed here, however, it requires the online presence of the Issuer.

An annex of [77] describes how to combine the Secure Electronic Transaction (SET) protocol[5.6] with EMV-compliant cards for electronic transactions conducted over the Internet. The complexity of the scheme is quite high. In addition, it requires the online presence

[5.6] http://www.setco.org

of a 'Payment Gateway' which is connected to the Acquirer's (and Issuer's) payment network. Finally, it is worth noting that Subscriber Identity Module (SIM) cards of mobile phones have recently been augmented with EMV-compliant applications[5.7] and that mobile equipment with EMV-compliant card readers has been available for some time.

5.7 Summary

In this chapter we have proposed a user authentication scheme for distributed systems that relies on EMV-compliant cards. These cards need to be able to perform asymmetric cryptographic functions (i.e. DDA) and must have a separate EMV 'Authentication Application' installed on them by their Issuers.

The CS itself acts as the AS for relying SPs. The scheme does not require online participation of the Issuer, and its security does not depend on CS integrity, as core functions are delegated to the trusted card. It leverages the existing EMV PKI and preserves user mobility and privacy, and can be regarded as an alternative to other smartcard-based user authentication mechanisms (such as the use of Subscriber Identity Modules, as described in chapter 4).

The associated SSO protocol requires only minimal user interaction, yielding a potentially seamless user experience and allowing several transparent re-authentications to occur within a given user/SP session.

[5.7] www.oberthurcs.com

Chapter 6

An authentication scheme based on trusted computing

Contents

This chapter presents two types of interdomain user authentication scheme that are based on 'trusted computing', i.e. on computing platforms that conform to the Trusted Computing Group specifications. Some of the material in this chapter has previously been published in [156] and will be published in [160].

6.1 Introduction

The Trusted Computing Group[6.1] (TCG) is an industry standards body that is currently developing specifications for trusted computing hardware building blocks and software interfaces across multiple platforms, including personal computers, personal digital assistants and mobile phones. A platform that conforms to these specifications is termed a 'Trusted Platform' (TP).

A potentially large number of applications can benefit from this technology [9, 160]. A comprehensive overview of the specifications (version 1.1) and potential usage scenarios is given in [9]. In particular, [9, p.255] proposes the use of TPs for user authentication in a corporate environment by delegating crucial functionality to the TP; this chapter elaborates on this subject and extends the approach to open environments. In particular, methods for constructing both pseudo-SSO and true SSO schemes based on TCG-conformant platforms are described.

The chapter is organised as follows. The next section contains a review of relevant TCG architectural components and security services. Sections 6.3.1 and 6.3.2 describe how to construct pseudo and true SSO schemes respectively, based on the use of TCG conformant platforms. Section 6.4 analyses the two schemes with respect to privacy, and section 6.5 discusses issues such as implementation costs and cross-platform mobility. Finally, section 6.6 concludes the chapter with a summary.

6.2 Review of TCG security services

This section introduces the components of the TCG specifications relevant here. For a full description the reader is referred to [196, 197, 198].

Every TP has a Trusted Platform Module (TPM) which is essentially a crypto co-processor that is augmented with extra functionality. The TPM is irrevocably bound to the platform's main hardware, e.g. soldered onto its main circuit board. The TPM never exposes certain data that is stored inside its non-volatile memory; external software can only ask the TPM to execute certain commands that may involve this data. These commands are known as

[6.1] www.trustedcomputinggroup.org

TPM capabilities. Certain capabilities, if used properly in orchestration, enable the platform to perform higher-level operations, known as *TCG services*. Three such services are relevant to this chapter, namely TPM Identities, Integrity Metrics, and Secure Storage.

6.2.1 TPM Identities

Every TPM has a unique RSA key pair stored in it, termed the *Endorsement Key* (EK), the private part of which never leaves the TPM. Thus, in principle, the TPM could use the EK to digitally sign messages. The verifier of such a signature, if given a copy of the public part of an EK that is known to belong to an authentic, i.e. TCG conformant, TPM, can establish whether or not the message originates from (i.e. was signed by) a conformant TP. As we see below, being able to establish this is crucial in the context of TCG applications.

Unfortunately, this naive method enables the verifier to uniquely identify the TPM that signed the message; this is unacceptable from a privacy perspective. To address this problem the TPM does not use the EK for the generation of signatures. Instead, the TPM can be instructed to generate an arbitrary number of random RSA keys, called *Attestation Identity Keys* (AIKs), which will be used for signing instead of the EK. AIKs effectively function as aliases for the EK and, since they are random, verifiers cannot link different AIKs belonging (or not belonging) to a particular TPM. In order for the system to work, verifiers need to be convinced that the AIK was indeed generated by an authentic TPM. The TCG has specified two fundamentally different mechanisms to achieve this, one in version 1.1 and one in version 1.2 of the specifications. In the following we briefly describe both.

In the technique specified in version 1.1 of the TCG specifications, the user needs to obtain an 'attestation' certificate from a third party, called the *Privacy Certification Authority* (PRV-CA), for every AIK that is generated by his TP. The certificate vouches for (or 'attests to') the fact that the AIK in question was indeed generated by a conformant TPM/TP. As the PRV-CA obtains a copy of the (public part of the) TPM's EK during this process, it can trivially link all the AIKs of any given TPM. In some scenarios this is, again, unacceptable from a privacy perspective.

The mechanism specified in version 1.2 [33] avoids this potentially undesirable property. The mechanism is developed from a particular type of cryptosystem known as an 'anonymous

credential system'[6.2]. This mechanism, called 'Direct Anonymous Attestation' (DAA), does not require an authority to certify each AIK. Instead, the TPM is required to obtain a digital 'attestation certificate' from an *Issuer* (typically the TPM or TP manufacturer) only once. This certificate can then be used by the TPM to digitally sign an arbitrary message in such a way that (a) the verifier is convinced that the message was generated by a TCG conformant TPM, and (b) the resulting signature cannot be linked to any other DAA signature that was previously generated by this particular TPM. By using this DAA technique to sign subsequently generated AIKs (instead of arbitrary messages) verifiers can be given assurance that these AIKs are genuine (i.e. that they were generated by a TCG conformant TPM), in a way that preserves their unlinkability, i.e. so that two AIKs associated with the same TPM cannot be linked.

The command that initialises the procedure by which a TPM obtains the attestation certificate from an Issuer is called `TPM_DAA_Join`. The command by which the TPM proves its authenticity to an external verifier and at the same time digitally signs a message is known as `TPM_DAA_Sign`. Simplified descriptions of these two operations follow.

`TPM_DAA_Join`**:** First, the TPM generates a 160-bit long-term secret f, which is never exposed outside the TPM. The TPM/TP then sends the (public part of its) EK to the Issuer and proves knowledge of f. The Issuer then verifies that the TPM in question is a genuine TPM. This can be done by comparing the public part of the EK to a list of keys that are known to belong to conformant TPMs. If the Issuer is satisfied that the TP/TPM is genuine, then it issues a Camenisch-Lysyanskaya (CL) [38, 137] signature on a 'blinded' version of f to the TPM/TP. This means that, while the issuer does not learn any information about f (apart from its length), the signature it returns to the TPM can be used to compute a valid signature on f. Prior to transmission to the TP, the signature is encrypted using the public part of the EK; as a result it can only be decrypted by the TPM it is destined for. On receipt of this encrypted signature, the TPM decrypts it, removes the blinding factor (i.e. computes a valid signature on f) and stores the result in its protected, non-volatile memory. This latter signature is the attestation credential; like f, its value is never exposed outside the TPM.

`TPM_DAA_Sign`**:** The TPM proves its genuineness to an external verifier by a zero-knowledge proof of knowledge (see section 1.1.5.4) of a CL-signature for the value f. The protocol is

[6.2] Anonymous credential systems are discussed in greater detail in part II of this thesis.

designed so that neither the value of f, nor that of the signature, are revealed to the verifier. In fact, no two protocol executions by the same TPM can be linked by the verifier, and hence use of the protocol does not reveal any connection to a specific TPM. The protocol is also designed so that the TPM can, at the same time, sign a message in such a way that a verifier can learn whether the message was generated internally (such as would be the case for an AIK) or input from outside.

Furthermore, the protocol enables the verifier to detect blacklisted TPMs, e.g. TPMs that have legitimately obtained attestation, but were subsequently compromised, i.e. had their secret value f and the attestation signature extracted. A compromised TPM would enable an adversary to convince a verifier that arbitrary messages originate from that TPM, even if this was not the case. The problem is addressed as follows: as part of the protocol, the TPM also computes a pseudonym N as $N = \zeta^f$, where ζ is a generator of a group in which the discrete logarithm problem is believed to be intractable. Although the verifier does not learn f, the protocol has the property that the verifier is given assurance that N is constructed using the same f as the one that was certified by the Issuer. If a TPM is compromised and its secrets become known, the value of f can be put on a blacklist; since the verifier learns ζ during the DAA signing protocol, it can compute $\zeta^{\hat{f}}$ for every $\hat{f}$ in the blacklist and compare the result to the value of N; if they are equal then the TPM has been blacklisted. Thus, the verifier can reject blacklisted TPMs using this mechanism. However, one should note here that protocol executions that involve the same value for ζ (and which therefore result in the same pseudonym N) can be linked by the verifier(s) as originating from the same platform; executions with different values for ζ remain unlinkable.

As mentioned above, the DAA protocol allows any message to be signed; the TPM could sign an arbitrary message by providing the (hash of the) message to the `TPM_DAA_Sign` command for signing. However, it appears that the specifications advocate a method that is more complicated but consistent with version 1.1. That is, the specifications suggest that the DAA-Sign protocol is first used to sign the public part of a given AIK, and the AIK is then used to sign messages. The reasoning behind this is that, if the verifier accepts a DAA signature as confirming that the originator is a genuine TPM, it will also accept messages that are signed by an AIK for which the public part has been DAA-signed.

6.2.2 Integrity Metrics

TCG-conformant platforms are able to reliably measure, store and report their software state. This is achieved by applying a one-way hash function to any software that is about to be executed on the TP; the resulting hash value (called an 'integrity metric') is stored inside the TPM's volatile, but protected, memory, in what are known as Platform Configuration Registers (PCRs). The initial value of a PCR is zero, but their values are updated during the TP's boot cycle. In particular, whenever the TP is about to execute a critical piece of software or firmware (such as the BIOS, the operating system and or an application), the following actions are performed:

1. The software about to be executed is cryptographically hashed (using SHA-1 — see section 1.1.2).
2. The resulting digest is concatenated with the current value of a specific PCR and this concatenation is hashed again.
3. The digest resulting from step 2 becomes the new value of the affected PCR.
4. An entry is appended to a log file called the 'history information'. The entry contains information about the measured event (such as the software name and version), which PCR was affected, and a 'validation certificate' (or a reference to it). The latter is a certificate issued and signed by the measured component's manufacturer or vendor that binds the component to the expected hash value of step 1.
5. The measured software is executed.

In this way, accurate 'integrity metrics' of the platform's software state are kept inside the PCRs, where they are protected from direct software interference. An external (communicating) party can assess the software state of a TP by issuing an 'integrity challenge' to the TP. This challenge includes a nonce that protects against replay attacks. Using the command `TPM_Quote`, the TPM generates a signature, using a user-selected AIK, over the current PCR values and some 'external' data that is input to the TPM. (This data typically includes the nonce.) The resulting signature, together with the public part of the AIK, the external data, the PCR values and the history information, is sent back to the external party as the 'integrity response'. Additionally, the TP has to convince the external party

that the AIK that has been used is authentic. How this is done depends on whether the version 1.1 or 1.2 mechanism is being used. For version 1.1, the PRV-CA's attestation certificate for the AIK's public part is sent to the external party, which verifies it using an authentic copy of the PRV-CA's public key. For version 1.2, the TP and the external party execute the DAA signing protocol; during execution of the DAA mechanisms, the TP signs the AIK's public part.

If the above process completes successfully, the external party is provided with assurance that the TPM is genuine, and that the AIK is associated with this TPM. It can then assess the trustworthiness of the TP's software state by

- verifying the TPM's signature over the PCR values and the external data (which includes the nonce) created using the AIK, and
- evaluating the history information and verifying that the given sequence of measured software components indeed yields the reported PCR values. This may include verifying the validation certificates issued by the manufacturers or vendors of the respective components.

If the above steps succeed, the external party can be confident that the TP's software state is as represented by the quoted integrity metrics (and has not been modified). Finally, it decides whether or not to trust this software state for the purposes of the communications. This decision is obviously subject to the external party's policy, and outside the scope of the TCG specifications.

6.2.3 Secure Storage

The TPM can generate an arbitrary number of RSA keys called 'storage keys'. The private parts of these keys are never exposed outside the TPM in unencrypted form; instead, prior to export from the TPM they are encrypted under a special 'root key' which only resides within the TPM's protected non-volatile memory. Any data that is encrypted under one of these storage keys can only be decrypted by the TPM that holds the corresponding private key. Every TPM-protected key, including storage keys and AIKs, has 'authorisation' data, a 160-bit string, associated with it. Knowledge of this string has to be demonstrated to

the TPM before any operation that makes use of the key, such as decryption, is performed. Additionally, storage keys can be configured to be bound to a particular software state. That is, the TPM can be configured to refuse to execute a command that involves any given key, unless the PCRs contain predefined values. Using this, combined with the authorisation data feature, it becomes possible to allow decryption of specific data only if the user demonstrates knowledge of certain authorisation data *and* the platform runs pre-determined and unmodified versions of a particular BIOS, operating system and application. We exploit this functionality for the schemes we propose in the remainder of this chapter.

6.3 Using Trusted Platforms for SSO

In this section, two methods for using a TCG-conformant platform to help build an SSO scheme are described. In particular, the following two subsections describe the construction of a local pseudo-SSO and a local true SSO scheme, both exploiting the presence of a TCG-conformant TP at the user site.

6.3.1 A local pseudo-SSO scheme

Recall that a potential disadvantage of pseudo-SSO schemes is that they do not remove the need for (securely) maintaining the user's SP-specific authentication credentials; this is because, in order to function, the AS needs access to them. TCG-conformant platforms can enhance the security of pseudo-SSO schemes by providing a secure repository for these credentials. The TPM service that is relevant in this scenario is that of secure storage, described above. We now describe how a pseudo-SSO scheme running on a TCG-conformant platform could work.

The AS is an application that the user runs locally. It behaves exactly like any other pseudo-SSO application: after locally authenticating the user, it automatically executes SP-specific authentication mechanisms on his behalf. In reference to Figure 6.1, this means that the automatically AS executes step 3 for as long as the user's authentication session, initiated in step 1, remains valid.

Using the secure storage capability, the AS is unable to decrypt the user's SP-specific cre-

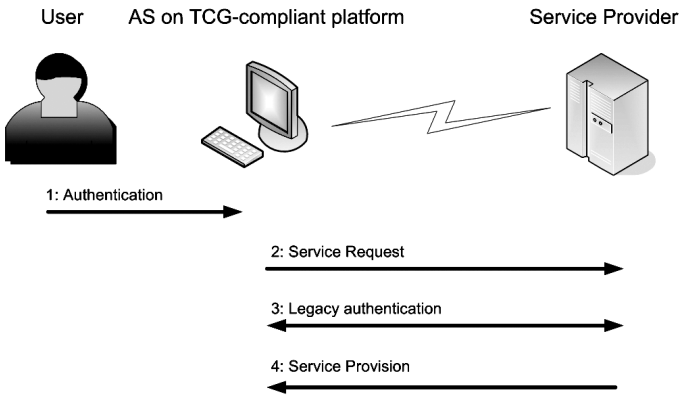

Figure 6.1: Local pseudo-SSO system based on TCG-conformant platform

dentials if the platform is not in a trusted software state; unauthorised alterations to the AS or the underlying operating system will raise an appropriate alarm. In particular, the system is configured so that the AS will decrypt the user's SP-specific credentials only if the following two conditions are met.

- The platform has booted into a trusted state.
- The initial user-to-AS authentication has been performed successfully.

There are two ways to guarantee that the second condition holds. The first is to completely delegate the initial user authentication and the management of the storage keys to the AS. If this is done properly, then only the AS can access the user's SP-specific credentials. A trustworthy AS would then access these credentials only after the user has completed the initial authentication successfully.

Another, perhaps more elegant, way to guarantee the second condition is by linking the user's initial authentication to the authorisation data that is required to retrieve the decryption key for his SP-specific credentials. If, for example, the initial authentication mechanism is based on a passphrase, the authorisation data that is required in order to access the storage key that is needed to decrypt the user's SP-specific credentials, could be a suitable one-way hash value (see section 1.1.2) of that passphrase. In this way, not only is the need for the AS to keep a copy of this data removed, but also other software that the user deems trustworthy can gain access to his SP-specific authentication credentials (for example, to perform a backup).

6.3.2 A local true SSO scheme

In a true SSO scheme, users and SPs have to trust the AS. For this reason, virtually all existing such schemes rely on an external trusted party that provides and controls the AS (e.g. [130, 144, 192]). Moreover, this party has to be online and available at all times. It would therefore clearly be advantageous to construct a true SSO scheme that removes the need for this trusted third party, or at least limits the amount of trust that users and SPs are required to have in it. With TCG-conformant platforms this can be addressed. In particular, although the need for an external party to *provide* the AS is not removed, the platform owner can be placed in *control* of it. This could be achieved in the following way.

The AS runs locally in the user machine, preferably as a continuously running process. As in the pseudo-SSO case, a session starts with the user authenticating himself to the AS. In this case, however, the SSO protocol also has to provide assurance to the SP that the AS can be trusted to behave appropriately, i.e. it will not deliberately issue false assertions. To this end, the protocol involves an integrity challenge/response message exchange during which the SP convinces itself that

- the platform is TCG conformant, and
- its software state is trustworthy, i.e. an AS that the SP trusts is running properly on the platform.

These conditions essentially guarantee to the SP that the software that is running on the user's platform (including the BIOS, the operating system and the AS) has not been modified, and that it is therefore likely to behave as expected. In particular, since a 'well-behaved' AS only executes the SSO protocol *after* the user has performed the initial authentication successfully, the integrity challenge/response mechanism guarantees to the SP that attacks that are based on a (maliciously) modified AS, for example a modification that simply skips the initial authentication, are prevented. In this scenario, of course, each individual SP rather than the platform owner selects which platform states it deems trustworthy, as part of its policy.

The AS, as part of the SSO protocol, also needs to convey an authentication assertion to the SP, i.e. a message containing the user's identifier and a description of his initial

authentication act. How this is done needs to be specified carefully. In particular, the proof of the integrity of the AS should not be conducted prior to sending the assertion, because this would enable the following simplistic impersonation attack. In such an attack, the adversary executes malicious software just after the integrity response is sent to the SP. This software then simply sends an assertion containing the identifier of any user without performing a genuine authentication process. Because the modification of the software state takes place after the verification of platform integrity by the SP (which we assume reflects a legitimate state), then the SP has no means to detect the attack.

Ideally, the assertion should be cryptographically bound to the verification of the AS in some way. As illustrated in Figure 6.2, this could be done by having the AS calculate a one-way hash value (see section 1.1.2) of the concatenation of the random number contained in the SP's integrity challenge with a hash value of the authentication assertion. The result should then be input to the TPM as the external data (i.e. the parameter `externalData`) of the `TPM_Quote` command that is executed during the integrity challenge/response session. (For a detailed description of this command see [197, p.116].) As long as the authentication assertion is sent to the SP (along with the integrity response), by reconstructing the hash chain the SP can verify that the correct random number was used in the calculation of the integrity response. The service provider must also carry out all the checks that correspond to the mechanisms used for the integrity response (step 3.4 in the figure), as described in section 6.2.1.

Note that the assertion that the AS constructs in step 3.3 does not contain an identifier for the user. The service provider identifies the user in step 3.4, as described below in section 6.4.1 (if the mechanisms of version 1.1 of the TCG specifications are used) and section 6.4.2 (if the mechanisms of version 1.2 of the TCG specifications are used).

6.4 Privacy

In this section certain privacy aspects of the two schemes described above are discussed. The pseudo-SSO scheme has, in principle, no significant differences, in terms of privacy, from other local pseudo-SSO schemes. Therefore, the discussion regarding local pseudo-SSO schemes in section 3.4.1 applies. We thus now focus on privacy aspects that apply to the true SSO scheme described in the previous section.

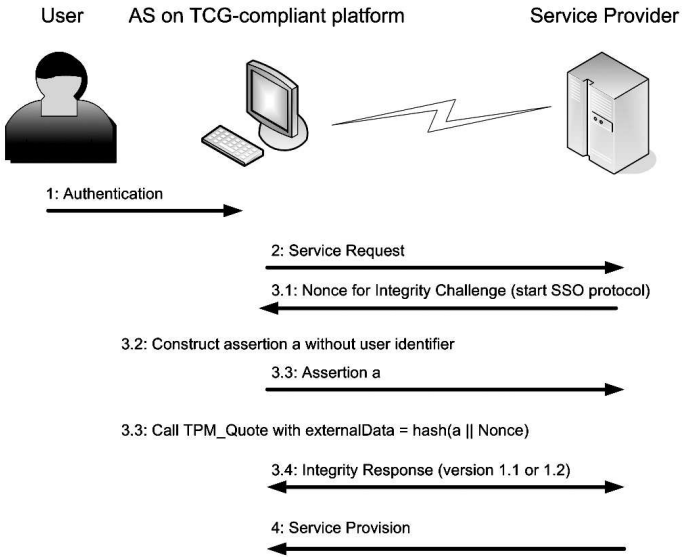

Figure 6.2: Local true SSO system based on TCG-conformant platform

As discussed earlier, most existing true SSO systems intended for use in an open environment depend on a trusted server. The operator of this server typically obtains access to certain information that might be considered private. This information disclosure is inevitable because otherwise the system would no longer work. According to the Liberty Alliance specifications [128, 129, 130, 131, 132, 133], for example, the operator of the server hosting the AS gets to know which SPs the users have a relationship with, their identifiers at those SPs, their login times, etc.

In contrast to these schemes, the scheme described in the previous section is under the control of the user. As no third party needs to be contacted during executions of the SSO protocol, the aforementioned information disclosure is avoided; the scheme is thus significantly more privacy friendly than existing server-based true SSO schemes.

However, as we have previously observed, if a user employs the same pseudonym with multiple SPs then, if these SPs collude, they can link user behaviour — this is clearly a potential breach of user privacy. We thus next focus on the case where users are known to different SPs under different pseudonyms. As discussed in section 3.4.1, a true SSO scheme has the potential to offer 'unlinkability' of these pseudonyms. In the following two sections, we examine the feasibility of meeting this requirement in the context of the TCG-supported true SSO scheme described in section 6.3.2.

As discussed in section 6.2.1, the provisions with respect to privacy have changed significantly between version 1.1 and version 1.2 of the TCG specifications. In particular, the way a platform is authenticated as conformant by a verifier (during an integrity challenge/response session between the platform and that verifier), is based on quite different cryptographic primitives in the two versions. Since the true SSO scheme described in section 6.3.2 above involves the TCG integrity challenge/response mechanism, its privacy properties affect the whole scheme.

6.4.1 Privacy under TCG 1.1

As explained in section 6.2.1 above, in version 1.1 of the TCG specifications, the TPM needs at least one AIK in order to be able to respond to integrity challenges. The AIK's public part has to be certified by a PRV-CA. The integrity response contains (among other things) the PRV-CA's certificate. The (public part of an) AIK can be used as a pseudonym in the SSO context, i.e. as an SSO identity. As the AIKs are randomly generated they do not reveal any information about each other, or the platform on which they were generated. They are thus unlinkable. Unfortunately, however, the process by which certificates for AIKs are obtained from the PRV-CA, allows the latter to unambiguously link (the public parts of) all AIKs that originate from the same TPM. This is because, during this process, it is required that the public part of the TPM's EK is revealed to the PRV-CA; this piece of information is unique to each TPM.

In practical terms this means that, by using platforms conformant to TCG version 1.1, it is possible to design a true SSO scheme where pseudonyms are unlinkable, *only under the assumption* that the PRV-CA does not reveal the links between AIKs (and therefore pseudonyms). That is, the level of privacy offered is limited by the degree to which a particular third party can be trusted.

6.4.2 Privacy under TCG 1.2

As described in section 6.2.1, the DAA sign protocol involves revealing a pseudonym $N = \zeta^f$ to the verifier every time that an integrity challenge/response protocol is executed. While these pseudonyms were included in DAA as a means to detect compromised TPMs, they can also serve as user pseudonyms in the SSO context. However, care has to be taken when

specifying how this should be done, as a DAA pseudonym is a function of data (i.e. ζ) that is either supplied by the platform alone, or by both the platform *and* the verifier of an integrity challenge.

An SP that requires a specific value for ζ to be used to construct the pseudonym, effectively prevents the establishment by the AS of more than one unlinkable pseudonym (for each attestation credential possessed by the TP). If a platform only has one attestation credential, in practical terms this means that the platform would be prevented from establishing more than one unlinkable pseudonym with that SP. Moreover, if the platform has multiple users, then the SP would be able to unambiguously link them.

It should be noted that users can obtain multiple attestation credentials for different TPM-generated f values, possibly from different issuers, and use different credentials for those pseudonyms that would otherwise be linkable through a common ζ value. However, the number of attestation credentials that a user may obtain is limited by the number of issuers that are willing to issue such credentials to a platform of the user's particular type. This is likely to impose a strict limit on the number of attestations that any given platform may obtain.

Colluding SPs that require the use of the same value for ζ may be able to unambiguously link pseudonyms corresponding to the same platform. Users who wish to maintain the unlinkability of their pseudonyms have to make sure that no SP requires, at any point in time, the same value for ζ as any other SP. Depending on the number of SPs and how often they change the value of ζ they require, this may be prohibitively complex.

Of course, this is a property inherent to DAA, and may also cause problems for other applications. As a result, possible techniques to avoid such a breach of privacy have been investigated elsewhere. One possibility, suggested in [33], is to derive the value of ζ from the SP's unique name using a collision-resistant hash function (see section 1.1.2). If done properly, this method prevents SPs with different names linking pseudonyms.

From a privacy perspective it would be desirable if the user was free to choose ζ in every execution of the DAA protocol. He could then choose the same value of ζ for those executions that should involve the same pseudonym, and thus be linked to each other, either for the same or different SPs.

More generally, and as observed by Camenisch [36], it can be argued that the level of privacy offered by TCG version 1.2 is, in certain circumstances, lower than that offered by TCG version 1.1. This is because, during execution of DAA (i.e. when using TCG version 1.2), the verifier learns the identity of the issuer that attested to the conformity of the platform. During a version 1.1 integrity challenge/response session, however, the verifier only gets to know the identity of the PRV-CA. As a PRV-CA would typically correspond to more than one issuer, the information that the verifier obtains during DAA (i.e. the identity of the issuer) allows it to identify the platform more effectively than it would be able to if it only learnt the identity of the PRV-CA. A scheme that equalises the level of privacy protection provided by DAA to that of the honest-PRV-CA scenario of version 1.1 is proposed in [36]. However, as this scheme introduces extra complexity (and potentially a third party), it remains questionable whether it will be used in practice.

6.5 Other issues

In this section we discuss a number of other issues that arise with respect to the two types of user authentication scheme described in section 6.3. The discussion is intended to complement, rather than replace, the discussion in section 3.4.

6.5.1 Significance of benefits

The TCG-based local pseudo-SSO scheme described in section 6.3.1 above offers certain security advantages over some of the other schemes in this category by offering hardware-based protection for the user's long-term authentication credentials. While this advantage is by no means insignificant, it is perhaps fair to say that, in general, a TCG-based true SSO scheme offers greater benefits to the user. This is because, as described in section 3.4, true SSO schemes have certain advantages over pseudo-SSO schemes. Moreover, the TCG-based true SSO scheme described in section 6.3.2 above, has the following two additional key properties.

Firstly, by using a TCG-conformant platform, the functionality of the AS can be delegated to the user platform. In many existing true SSO schemes this functionality necessarily resides on an external trusted third party device, because of the need for the SP to trust

the veracity of assertions made by the AS. In our scheme, although the AS software, and the hardware on which it executes, must nevertheless be manufactured and provided by trusted third parties, the user remains in control of it, although the user is, of course, unable to modify the AS without being detected. Secondly, the migration of the AS to the user platform leads to decentralisation. In contrast to existing true SSO schemes, where the party that controls the AS constitutes a central point of failure (and is therefore susceptible to service denial attacks), a scheme of the above type effectively distributes the centralised functionality among all users; the scheme is therefore decentralised, robust and scalable.

6.5.2 Man in the middle attacks

Both types of system only offer user-to-SP authentication. They are therefore vulnerable, like the scheme described in chapter 4, to the trivial man-in-the-middle attack described in section 4.4.4. In many situations it is thus necessary for the user to authenticate the SP before the SSO protocol is executed. This requires the SPs to have unambiguous identifiers that the user must manually inspect. This requirement can be met using well-established methods, such as SSL/TLS.

6.5.3 Cross-platform mobility

Cross platform mobility refers to a user's ability to use a scheme from more than one platform, using the same set of pseudonyms. Hence, in an SSO scheme that offers cross-platform mobility, the user's pseudonyms are independent of the particular platform he is using. In other words, if a scheme offers cross-platform mobility, then the AS that is invoked when the user is using, say, his office machine, logs him into SPs using the same set of pseudonyms as the AS that is invoked when he is using, say, his home computer.

In the context of the scheme described in section 6.3.1, it is conceptually easy to enable cross-platform mobility. This primarily involves copying the user's SP-specific authentication credentials from one platform to the other. Although the scheme we have described requires them to be encrypted under platform-dependent keys, the AS could be provided with a special 'migration' function that allows for the necessary key transfer, and associated decryption and re-encryption, to take place. The key migration mechanisms that are described in the TCG specification could be used for this purpose.

In true SSO schemes where the AS resides on a trusted third party, cross-platform mobility is an inherent feature. Unfortunately, the situation is different for the scheme described in section 6.3.2 above, in particular if pseudonyms are derived from AIKs or DAA pseudonyms (as described in sections 6.4.1 and 6.4.2 respectively). Because AIKs and some of the factors used to compute DAA pseudonyms are strictly platform-dependent and non-migratable, so will be the user's pseudonyms; different platforms necessarily involve different pseudonyms during the integrity challenge/response protocol.

Nevertheless, it is still possible to achieve cross-platform mobility. This, however, needs to be implemented as a separate service 'on top' of the scheme; such a service will allow a user to explicitly link different pseudonyms at an SP. Unfortunately, this solution not only appears to be somewhat unnatural in the TCG context, it also needs to be specified and implemented carefully in order to avoid introducing new attacks. It would appear that, under true SSO using TCG-conformant platforms, there exists a tradeoff between the potential unlinkability of pseudonyms and cross-platform mobility.

6.5.4 Complexity of managing trusted states

Under pseudo-SSO, it is the platform owner who classifies software states as trustworthy. Since the verification of whether the platform is actually in one of these states only occurs locally, the associated management overhead is not particularly complex. In the simplest case, the platform owner may just record the integrity metrics of a new installation (i.e. one that he is confident to be flawless), and specify this is the only trustworthy state.

Unfortunately, the situation is different for true SSO. Here, SPs need to be able to distinguish the software states that they trust from those that they do not. Depending on how the platform's software stack (BIOS, OS and other applications) reports integrity metrics to the TPM, this may be a rather complex task. In order to see why this might be so, consider the following scenario.

An SP that trusts the AS manufactured by some company X, decides to classify as trustworthy platform states that reflect a running instance of this particular AS. The AS comes in two different versions, AS_A for operating system A and AS_B for operating system B. Suppose that users of TCG-conformant platforms that run A and B, install AS_A and AS_B

respectively. Assume also that the platforms actually come with, say, 15 different BIOS versions (or that some users choose to update their BIOS). Furthermore, assume that A is an open-source system that users are free to modify. So, the users who run A, actually run slightly different, although perfectly legitimate, versions (or 'distributions') of it, say A_1 to A_{18}. The problem is that, each combination of BIOS, operating system and AS yields a different set of integrity metrics, even if individual components differ only slightly. In the above example, the SP would have to be aware of $15 \times 18 + 15 = 285$ different software states and classify them as trustworthy, just to be able to recognise acceptable software configurations that include the one AS it has chosen to trust. Imagine what will happen if the AS manufacturer periodically releases new versions of the AS and some users still use older versions. The number of integrity metric sets that represent equally trustworthy software configurations increases at an exponential rate as the number of measured software components grows. Although some efforts have been made to reduce this rate (by grouping the TPM's PCRs together according to functional categories), it remains unclear how complex, or practical, the management of trusted states is going to be in practice.

6.5.5 Costs

The general discussion in sections 3.4.5 and 3.4.6 about the deployment, maintenance and running costs of interdomain user authentication schemes also applies to the two schemes in this chapter. This section therefore focuses only on the relevant implementation costs.

Implementing applications that make use of TPM functionality is, at the time of writing, much more expensive than implementing 'traditional' software. This is primarily because the implementation of TPM-aware BIOSs and operating systems is still under development.

Even if an implementer manages to properly interface with the TPM, the current lack of integration and support for reporting integrity metrics at the BIOS and operating system level hinders testing in a different environments, i.e. on platforms with different BIOS or operating system versions.

When contrasting the implementation costs of the two schemes described in section 6.3, it appears that, generally speaking, the pseudo-SSO scheme of section 6.3.1 is cheaper than the true SSO scheme of section 6.3.2. This is because the latter is more complex than the

former; as it includes the integrity challenge/response mechanism, its testing phase requires the cooperation of multiple, preferably different, TCG-conformant platforms.

6.5.6 Open source software

It is sometimes said that open source software is, in general, more trustworthy than software for which the source code is not publicly available. This, it is said, is because the source code can be reviewed by security experts who can point out weaknesses. Unfortunately, experience has shown that potential attackers can also do so. In practise, open source software is not necessarily less flawed than closed source software. In fact, commercial software (which is typically not open source) is sometimes supported by contractual agreements that settle liability issues in case of security incidents. In these cases, the coverage of costs incurred through the exploitation of security-related software flaws may be settled by legal means.

However, the situation is perhaps different in the context of the SSO schemes described in this chapter. In section 6.3.2, in particular, we argued that TCG conformant platforms enable the user, as opposed to some third party, to be in control of the AS. Although this may be true, in order for this control to be essential, we argue here that it is advantageous for the AS's source code to be publicly available. This is because it can be established, via independent means, how exactly the AS is doing what it claims to be doing, thereby enabling users to exercise their control more effectively.

Public availability of the AS's source code, may be even more advantageous in the context of privacy, as discussed in section 6.4; users who wish their pseudonyms to remain unlinkable are likely to require some form of assurance that the AS does not disclose (at least in any obvious way) any unnecessary information. Although some may trust the AS provider to properly implement the SSO functionality, this does not necessarily imply that they trust the AS provider to protect their privacy as well. As it is easy to hide information leakage in closed source software (thereby avoiding the raising of alarms), it would appear potentially easier to convince a wary user that an open source, as opposed to closed source, AS will effectively protect their privacy.

6.6 Summary

In this chapter it was shown how interdomain user authentication schemes may be built using computing platforms that are conformant to the TCG specifications. The construction of both pseudo and true SSO systems were considered, as well as the use of both available versions of the TCG specification. The proposed systems make use of three security services, namely Secure Storage, TPM Identities, and Integrity Metrics. User authentication is delegated to the local TP. Under the true SSO variant, SPs need to check the authenticity of the user's TP and the integrity of its software state before trusting authentication assertions.

The schemes are independent of the local user authentication method and protect user privacy through the use of pseudonymous SSO Identities. However, as these identities are bound to the platform under true SSO, this variant does not support cross-platform user mobility per se. Furthermore, the inherent complexity of the TCG specifications is inherited.

Chapter 7

A user authentication scheme suitable for use from untrusted devices

Contents

This chapter presents the design of a proxy-based interdomain user authentication scheme that is suitable for use from an untrusted network access device. Unlike existing proxy-based SSO schemes, this scheme does not require the proxy and the service providers to share a security infrastructure. An open-source implementation of the scheme, called 'Impostor', is also described. The prototype is implemented as an HTTP proxy, resulting in a system that works with common web browsers. The source code of Imposter can be found in Appendix A. Much of the material in this chapter has previously been published in [159].

7.1 Introduction

One advantage of proxy-based SSO systems is that they have the potential to enable user authentication to a variety of remote service providers from untrusted network access devices in a way that does not compromise the secrecy of long-term user authentication credentials. This applies even when the authentication methods used by remote service providers would normally not offer any protection to user credentials, e.g. the use of passwords. This chapter presents the design of a pseudo-SSO scheme with this feature.

The chapter is organised as follows. The next section contains a definition of what we mean by an 'untrusted' network access device, and section 7.3 presents the objectives, architecture and properties of the scheme at a design level. Section 7.4 presents 'Impostor', a concrete prototype implementation of the scheme in the form of an HTTP proxy. Section 7.5 discusses additional technical issues, and sections 7.6 and 7.7 give an overview of related work and a summary of the chapter.

7.2 What is an untrusted network access device?

A network access device can compromise security in a number of ways; it can log and disclose all its internal state (including sent and received messages, cryptographic keys and passwords), it can spoof (both local and remote) interfaces, and it can simply deny service. While users do not expect a trustworthy device to engage in such malicious behaviour, the meaning of a network access device being 'untrusted' has to be precisely defined and assumptions about it made explicit in order to provide a clear understanding of the scheme's objectives.

In order to clarify matters, consider the following scenario, around which the scheme is built. Suppose a user needs to access a network service, say an e-mail account, from a network access device that he does not particularly trust, for example a public terminal at an airport or an Internet café. The user is typically willing to trust the terminal to act as the communication endpoint and input device during this particular online session, but for no longer. Now suppose that the e-mail provider authenticates users using long-term credentials, such as username/password pairs. Providing these long-term credentials to the

public terminal, however, constitutes a serious risk, since knowledge of these credentials enables the terminal (and its operator) to impersonate the user to the service provider, even after the session has ended. The primary purpose of the scheme described here is to avoid this risk.

Based on the above scenario, a network access device is deemed 'untrusted' if the following conditions hold.

- There exists a temporary trust relationship between the user and the network access device: the former trusts the latter only for the time period of a particular session, but for no longer. In particular, during that session, the device is trusted
  - not to spoof local and remote user interfaces, and
  - not to hijack communication sessions with individual services.
- The device is not trusted to protect any long-term secrets, i.e. secrets that remain valid *after* a particular online session has completed, and that may be entered or otherwise made available to it.

Issues that arise if these conditions are relaxed are briefly discussed in section 7.5.1. It is worth noting that the use of secure channels, such as those provided by the Secure Socket Layer (SSL) or the Transport Layer Security (TLS) protocols [180], does not help to prevent sensitive data being made available to the untrusted device, although a secure channel would offer protection for exchanged data against third party interception and/or manipulation.

7.3 Description of the SSO scheme

This section describes the objectives, the architecture and certain key properties of the SSO scheme at a design level.

7.3.1 Objectives

Methods exist that enable users to authenticate themselves to remote entities without having to disclose long-term secrets to the access device they are using. Examples include

challenge/response protocols and one-time passwords, see, for example, [142], and products based on such ideas are widely available. These authentication mechanisms are suitable for use from an untrusted device, but, unfortunately, are rarely used by providers of public network services. The majority of services authenticate users based on a username and password.

Based on these observations, the three primary objectives of the SSO scheme are as follows.

1. The scheme should enable users to authenticate themselves from an untrusted network access device (as defined in section 7.2) to remote service providers whose login mechanisms involve long-term secrets (such as passwords), without, however, disclosing these long-term secrets to the access device.
2. The scheme should require users to manage only one set of authentication credentials, thereby providing SSO.
3. The scheme should be transparent to network service providers; they should not even need to be aware that it is in place.

7.3.2 Architecture

The architecture of the scheme is based on a trusted SSO proxy that keeps a copy of the user's long-term authentication credentials in a suitably protected credential database. All network traffic between the untrusted device and the remote service providers is physically routed though that proxy. The information flow that occurs in the event of a user trying to log into a network service is depicted in Figure 7.1. As shown in that figure, the user's login request to a remote network service provider is recognised and intercepted by the proxy in step 1. In step 2 the proxy authenticates the user using a suitable one-time authentication mechanism, typically including a challenge issued by the proxy (step 2.1) and a response from the user (step 2.2). Assuming successful user authentication, the proxy executes the authentication mechanism used by the network service on behalf of the user in step 3. This would typically involve employing the user's long-term secrets that are stored in the credential database. The secrets, however, never reach the untrusted network access device, as step 3 occurs directly between the proxy and the remote service provider. Finally, assuming that step 3 completes successfully, the service is provisioned in step 4 and the first objective listed in

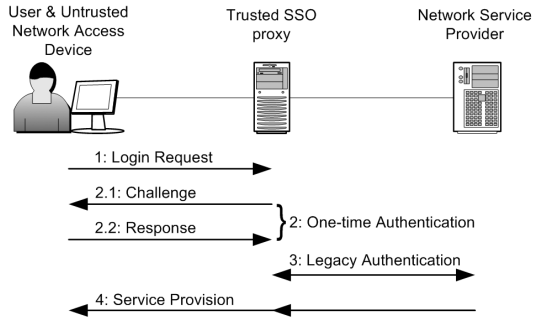

Figure 7.1: Architecture

section 7.3.1 is met.

The second objective is met by differentiating between users (in step 2) based on a unique identifier which is decoupled from any particular network service; in order to use the system, users only have to memorise or otherwise have available that single identifier, and to carry out the one-time authentication procedure with the proxy. Finally, the third objective is met by having the proxy replace the user's network address with its own when forwarding requests to network services. In this way service providers only 'see' the proxy and are therefore not aware of step 2 taking place. According to the taxonomy of chapter 3, this architecture falls into the category of proxy-based pseudo-SSO schemes.

7.3.3 Properties

As a proxy-based pseudo-SSO scheme, the system inherits all the properties of that particular class of SSO scheme. As discussed in section 3.4, changes in a network service's proprietary authentication interface have to be reflected at the proxy. This means that maintenance costs may be relatively high, especially in a dynamically changing environment. The fact, however, that the proxy is transparent to service providers enables its immediate and cost-effective deployment.

As is the case with any proxy-based SSO scheme, the proxy constitutes a single point of failure and has to be protected against service denial attacks. Its credential database has to be properly protected against illegal access, as it contains long-term secrets that allow impersonation of users to all SSO-enabled network services. However, this is not as severe an assumption as it might seem; in practice the proxy might be implemented on the user's own

'home PC', which would already typically be trusted to store such secrets (e.g. in the form of cookies), and is not likely to be a major target for service denial attacks. In the corporate setting, on the other hand, using proxy chaining and load balancing techniques [10] could offer an acceptable level of resistance against service denial attacks. Moreover, the credential database could be stored in an encrypted form at a centralised server within a protected area of the intranet.

As a consequence of the proxy's need to intercept all traffic between the user and the rest of the network, no end-to-end secure channels between the untrusted device and any service are allowed; if a secure channel is nevertheless required at the application level, the proxy has to set up two separate secure channels, one with the user and one with the service, such that it can carry out its functions.

7.4 The Impostor prototype

This section presents 'Impostor', a proof-of-concept implementation of the SSO architecture described in the previous section. Although the SSO architecture is generic in the sense that it could be used for several network services, such as the File Transfer Protocol [171] or the Simple Mail Transfer Protocol [101], Impostor is realised as an HTTP (web) proxy. Thus, the prototype offers SSO for websites only. It is intended to be started as a continually running process (also known as 'service' or 'dæmon') on a trusted 'SSO server' machine. The implementation was written in Java 1.4 and has been successfully tested with a variety of common web browsers and operating systems, both Unix-based and Windows-based. It is available as an open source project at `http://impostor.sf.net`.

The core of the software consists of an HTTP proxy, augmented with the SSO functionality described in the previous section; every incoming HTTP request to the proxy is analysed (step 1 in Figure 7.1) and, if it is recognised as a login request for a website for which the proxy has been equipped with user authentication credentials, the proxy presents the user with a (customisable) login web-page that contains a freshly generated challenge, and asks the user to provide his/her unique identifier and a response (step 2). If the identifier is valid and the response matches the challenge, the user's long-term credentials for the website in question are filled into the initial HTTP request from the user, which is then forwarded to the website (step 3). Otherwise, an error page is returned. If an incoming HTTP request is

not recognised as a login request then the proxy simply forwards it to the website.

It is worth noting that the proxy rejects replays of previously accepted responses and responses that are received after a specified timeout period. This is necessary because each HTTP request/response is carried over a separate lower level connection. This means that the Impostor login page, sent to the network access device as an HTTP response and which contains the challenge, is carried over a connection that is separate from the one that carries the user's response to the challenge (which is a new HTTP request). Therefore, it might be possible for an active adversary, who observed a valid challenge/response pair, to replay that response within the specified timeout period. This attack is prevented because Impostor would reject the replay.

An alternative way to implement the Impostor proxy would be to have it analyse every HTTP *response* and to check whether it is a login page for a SSO-enabled website. This approach would potentially make the user experience more seamless, since the user would not be required to submit an 'empty' login request to the website first. However, the decision to have the proxy analyse HTTP requests, rather than responses, was taken for the following two reasons. Firstly, login pages typically provide a number of login-unrelated links that the user may want to follow. It would therefore potentially significantly impair usability if the scheme were to presume that the user wanted to log in every time that the user visited a page providing the possibility of performing a log in. Secondly, the layout and content of login pages tends to change more often than the format of the actual login requests. Also, websites may offer multiple login pages that nevertheless yield a common login request format. The requirement for the proxy to recognise multiple and changing login pages would reduce its efficiency and increase its maintenance costs.

If, at any stage, it is necessary to set up an SSL/TLS channel between the user browser and a web server (via the HTTPS scheme)[7.1], the proxy sets up two separate channels: one with the website and one with the user browser. The certificate that the proxy uses to set up the browser-side SSL/TLS connection does not, of course, match the website's certificate, but, fortunately, this does not disrupt the service as browsers typically offer the option to accept 'unknown' certificates via a simple yes/no dialogue[7.2]. Moreover, users should expect this,

[7.1] An SSL/TLS channel might be required for a variety of reasons. Examples include the user directly typing into his browser the address of a web page that is served using SSL/TLS, or clicking on a link to such a page. The main reason that login pages are served over SSL/TLS is in order to protect against password disclosure via eavesdropping.

[7.2] The situation is fortunate in the context of an untrusted network access device and the scheme described in this chapter; in other scenarios it is certainly a weakness [153].

as they will certainly know that Impostor is being used, and will also know the web page via which the proxy is operating.

Impostor offers an additional privacy protection service by passing every web page that is sent from the web server to the network access device as part of an HTTP response to a method that may filter its content, before forwarding it to the browser. In this way it is possible to prevent personal information, such as a user's address or credit card details, ever reaching the untrusted device. It is worth emphasising that this filtering service is not disrupted in the presence of an SSL/TLS channel.

The software is designed in a modular fashion, so that it is easy to extend or replace individual components without affecting the functionality of unrelated components. In Java terminology [81], the core proxy dæmon classes form a *package* that 'talks', via well-defined APIs (i.e. Java interfaces), to the supporting components (i.e. classes) that are responsible for (1) recognising and 'filling in' HTTP requests for websites, (2) performing the one-time authentication mechanism, (3) managing the credential database, and (4) performing content filtering. In other words, the proxy dæmon is logically decoupled from these four components; it does not directly implement their functionality. Thus, separate *implementations* of these interfaces (as given in Appendix A) need to be provided in order to make Impostor work. The interfaces, along with their current implementations, are briefly introduced in the remainder of this section.

7.4.1 The RequestRecognizer interface

The RequestRecognizer interface provides access to the methods supporting the identification and processing of user login requests to websites. Every incoming HTTP request is passed to a method (called `init`) of this interface. If the implementation recognises it as a login request to a website (step 1 in Figure 7.1), the proxy is informed (via the `isRecognized` method) and invokes the SSO protocol, as explained above. The implementation of this interface must also provide a method that 'fills in' user credentials (i.e. usernames and passwords) into the request. The proxy calls this method (`fillInUsernameAndPassword`) after successful one-time user authentication (step 2) in order to perform the service-specific legacy authentication (step 3). If an incoming HTTP request is not recognised as a login request then the proxy simply forwards it to the website.

The current implementation recognises and 'fills in' login requests for three web-based email providers, namely two well-known public providers and a university service.

7.4.2 The ChallengeResponseManager interface

The proxy uses an implementation of the ChallengeResponseManager interface in order to perform the one-time user authentication; the interface provides methods for issuing a new challenge (`getNewChallenge`), determining whether or not a given user identifier is valid (`isValidIdentifier`), and verifying a given response to a previously issued challenge (`verifyResponse`).

The challenge/response mechanism of the current implementation requires users to share a passphrase (at of least eight characters) with the proxy server. The challenge/response mechanism consists of the proxy asking the user to provide three randomly chosen characters from his passphrase (e.g. the second, fifth and last). If the user fails to provide the correct response, the server keeps asking for the same set of characters until either the correct response is given, or a maximum number (e.g. five) of failures has been recorded. In the latter case, the user's account at the proxy is disabled. This challenge/response mechanism could be, in principle, easily replaced with any other mechanism, such as one of those mentioned in section 7.3.1.

7.4.3 The UserManager interface

The UserManager interface defines the API used by the proxy dæmon to interact with the credential database management component. The implementation must provide functionality that maps proxy users to their website-specific authentication credentials. In particular, methods must be provided that retrieve the website-specific username (`getUsernameForIdentifier`) and password (`getPasswordForIdentifier`) for any valid proxy user identifier/SSO-enabled website combination.

The current implementation uses a simple, text-based credential database, but this could be replaced by a more sophisticated scheme.

7.4.4 The ContentFilter interface

The ContentFilter interface provides methods used to support the privacy protection service mentioned above. These methods filter HTTP response headers (`filterHTTPHeaders`) and web pages (`filterWebPageLine`) before the proxy forwards them to the browser; the implementation looks for any information that should not be sent to the untrusted network access device and substitutes it with a neutral (or empty) string.

The current implementation looks for a set of strings specified by the administrator. If any of these 'sensitive' strings is found, it is replaced by a string that is chosen randomly from a set of substitution strings defined for that specific sensitive string. This fairly basic (but nevertheless effective) filtering mechanism could be replaced with a more sophisticated content filtering technique.

7.5 Other issues

7.5.1 Relaxing the assumptions

As explained in section 7.2, the untrusted network access device is assumed not to spoof user interfaces or to hijack communication sessions with individual services. This section briefly examines the implications of relaxing these assumptions.

If the untrusted device spoofs the user interface then there is no assurance that the user is communicating with the entities he believes he is communicating with, even in the absence of external attacks. This is because authenticating a remote entity requires an interface that informs the user whether or not authentication was successful. If this interface is provided by a trusted device, such as a trusted smartcard reader, authentication of services (including the SSO proxy) could be supported in this untrusted environment. The value of such an authentication procedure, however, is undermined in the presence of session hijacking. If the untrusted device hijacks the user's session (either on its own or by colluding with another entity) it can effectively abuse the user's authentication status at any service. This renders any authentication mechanism (unilateral or mutual) between user and services useless, even if it results in a key that is supposed to cryptographically protect subsequent communications (as is the case with SSL/TLS channels).

It is clear that, if the untrusted device spoofs its interfaces or hijacks the communication sessions with individual services, then the user's entire online session is compromised. The threats posed by a network access device that engages in such malicious behaviour are not addressed by the scheme described here. It is nevertheless worth noting that, even in the presence of interface spoofing and session hijacking, users of the scheme described here do not have to disclose long-term secrets to the untrusted device; moreover, as long as those secrets are not disclosed by other sources, only the current session is compromised.

7.5.2 Some technicalities

This section documents certain technical issues that arose during the implementation of the SSO proxy.

7.5.2.1 Setting up Impostor

As explained in section 7.4 above, the Impostor prototype is independent of its supporting components. Therefore, 'setting up' the system mainly involves setting up the components responsible for the one-time user authentication mechanism, and performing credential database management, i.e. populating it with the user's Impostor and website-specific authentication credentials. It is also necessary to generate the asymmetric key pair (and a corresponding public key certificate) that Impostor will use when setting up browser-side SSL/TLS connections. Finally, the Impostor login webpage and error page have to be designed according to some simple guidelines that allow the proxy to dynamically insert challenge values (and other details) at runtime.

7.5.2.2 HTTP Authentication

In [83] the "HTTP Authentication" method is specified; this method is sometimes used by websites and web proxies to authenticate users. Although there is provision for a challenge/response mechanism with the one-time property (called 'Digest Access Authentication'), it involves users typing their usernames and passwords into the access device. Thus, since use of this method will not provide any additional security, during user-to-proxy authentication (step 2 in Figure 7.1), a standard web form is used to acquire the user's identifier

and response.

By contrast, the SSO proxy can perform the HTTP Authentication protocol to authenticate users to websites (step 3), as long as the RequestRecognizer component supports it. (In fact, the university web-based email service mentioned in section 7.4.1 uses HTTP Authentication.)

7.5.2.3 The use of cookies

The web proxy does not, by default, interfere with the cookies that individual services may store in the browser. It is possible, however, to filter out cookies using the content filtering facility described above. The proxy itself does not save cookies in the browser.

7.5.2.4 Persistence of connections

Currently, the proxy does not support persistent connections; every incoming HTTP request is processed independently of others. This has a number of side effects: on the one hand, the user has to re-authenticate every time the proxy is asked to provide long-term credentials; this reduces the exposure to session hijacking. On the other hand, the performance is degraded in certain circumstances.

7.5.2.5 Real world applicability

From a practical point of view, the main drawback of the Impostor prototype is that users are required to configure their browsers to use the proxy, i.e. to route all web traffic through the proxy. This implies changing the settings of the untrusted device's browser. Some public terminals impose restrictions on the ability of users to make such a configuration change. Another issue may arise if firewalls are located between the untrusted device and the proxy. Running the proxy on a port that is usually not blocked by firewalls (such as port 80 or 22) may help circumvent that problem.

7.6 Related work

Web-based SSO schemes that are based on a trusted SSO proxy, such as the Liberty Alliance [130] specifications and Microsoft Passport [144], are probably the most closely related schemes to the Impostor prototype. Liberty, in particular, does not require the use of any specific user authentication method. This means that, if a suitable mechanism is selected, use from an untrusted network access device can be supported. However, as pointed out in Chapter 3, Liberty and Passport are true SSO schemes. This means that explicit relationships between the SSO proxy and the service providers need to be established and supported by a (potentially costly) common security infrastructure (such as a Public Key Infrastructure) that spans the SSO proxy, all participating websites and possibly also the end-users. By contrast, as the scheme described here is transparent to service providers, it does not require explicit relationships or a security infrastructure between the proxy and service providers; the solution is far more flexible and easily deployed. Furthermore, it can be implemented on a small or large scale; the proxy can be operated by any organisation, including individual users, and not just by organisations with well-established relationships with service providers.

Proprietary local SSO schemes, such as Novell's SecureLogin[7.3], Passlogix' V-GO[7.4] and Protocom's SecureLogin[7.5] are SSO solutions that can be deployed transparently to service providers. The fundamental difference between these systems and the prototype described here is that these systems are not suitable for use from an untrusted network access device; the local device necessarily gets access to the user's service-specific authentication credentials.

Another area of related work is that of content filtering proxies. The majority of existing content filtering proxies focus on the protection of minors from inappropriate content and the imposition of restrictions on employees in a business environment. Although the purpose of the content filtering functionality of the system presented here is different, the concept is similar. However, one ought to keep in mind that a malicious end-user device can always 'switch off' the proxy without notification.

[7.3] www.novell.com/products/securelogin
[7.4] www.passlogix.com/sso
[7.5] www.protocom.cc

7.7 Summary

This chapter presented the design of a proxy-based pseudo-SSO scheme and described a prototype implementation. The scheme essentially overlays a single, one-time, authentication method over the existing authentication mechanisms of individual service providers, thereby providing SSO. The prototype, which works 'in the real world' as an HTTP proxy in a manner completely transparent to websites, is easily extensible and does not depend on any particular one-time authentication method or database component; it may be deployed by individual users or in a corporate setting.

Looking into the future of ubiquitous computing, where personal devices with limited user interfaces might need to connect to various service providers as users roam, pseudo-SSO schemes such as the one presented in this chapter might prove useful; the limited devices are only required to perform a single authentication mechanism with the trusted proxy. The proxy, as a middleware service, then executes predefined authentication mechanisms with each individual service provider, not only circumventing the (costly) need for system-wide agreements and infrastructures, but also transparently adapting to a dynamically changing environment.

Part II

Anonymous Credential Systems and Single Sign-On

Chapter 8

Anonymous credential systems and timing attacks

Contents

This chapter gives an informal introduction to anonymous credential systems, presents a short literature review, and points out that there exist inherent limits to the privacy that one can obtain by using such systems. These limits are then used in order to describe a series of timing attacks that can be launched against such systems. These attacks aim at reducing the degree of unlinkability of events within the system, thereby adversely affecting the anonymity a user might enjoy. Much of the material in this chapter has previously been published in [161].

8.1 An introduction to credential systems

A credential system enables subjects, typically human users, to prove possession of attributes to interested parties. These parties are usually referred to as *verifiers* as they verify whether or not the subject possesses the attributes of interest. A simple real-world example of a credential system is theatre ticketing. A theatre visitor is typically required to produce a valid ticket when requested by an inspector. The credential in this example is the ticket

itself, while the visitor's eligibility to attend the performance is the attribute. The visitor's right to this attribute is demonstrated by showing the ticket (i.e. the credential) to the inspector.

In general, users of a credential system need to obtain the credential from an entity termed the credential *issuer*. The issuer encodes some well-defined set of attributes together with their values into the credential which is then passed on, or 'granted', to the user. The user subsequently 'shows' the credential to a verifier in order to obtain a good or service, where the term 'show' does not necessarily mean that the credential is passed to the verifier, but, more generally, it implies that the user proves to the verifier that it possesses a credential with the relevant attributes.

Issuers are typically well-known organisations with authority over the attributes they encode into the credentials they issue (for example a theatre ticket box office), and verifiers typically are service providers that perform attribute-based access control (for example a porter that controls admission to a theatre hall).

The credential system is the infrastructure and set of procedures by which the players in the system interact. There are several desirable properties for such a system. For instance, in the example above, it should not be easy for anyone to obtain a ticket by any means other than by purchasing it from the box office. In other words, it should not be possible for anyone to easily forge a credential. This is why theatre tickets are sometimes printed on a special paper that is hard to forge.

In general, the user of a credential system should be able to show a credential, i.e. prove possession of it only after having gone through the process of obtaining it from the legitimate Issuer. Furthermore, the user should be able to prove possession of only these attributes that were encoded into the credential by the Issuer, even if the user colludes with other entities that may have legitimately obtained credentials that encode other attributes.

In the digital world, credentials have to be finite sequences of bits. An example of a digital credential system is a Public Key Infrastructure (PKI) (see section 1.1.5). In a PKI, credentials are public key certificates that bind together attributes of the subject such as name, public key, its issue and expiry dates, etc. In this case the credential issuer is a Certification Authority (CA) that grants public key certificates after following a user registration

procedure. Credential verifiers are the entities within the PKI that accept the certificates issued by the CA and are often known as the *relying parties*. The measure that protects credentials from being forged in this system is the inclusion of the CA's digital signature in every certificate, and the fact that this signature is checked for correctness by the verifier.

8.1.1 Anonymous credential systems

In a conventional credential system, for example a PKI, issuers and verifiers will typically identify any given subject using a system-wide identifier. This has a potentially severe impact on the subject's privacy, as it enables issuers and verifiers to combine their knowledge about the behaviour of the subject. Indeed, they can construct individual transaction histories for all the subjects in the system, simply by correlating the issuing and showing of credentials using these identifiers.

In certain applications, e.g. electronic cash, this privacy issue is significant. In such applications it is often desirable to prevent issuers and verifiers from being able to correlate credential-related events. This is obviously not possible if a global identifier, such as a subject's name, is disclosed to issuers and verifiers during the issuing and showing of credentials. Credential systems that avoid disclosure of a global user identifier to issuers and verifiers are termed *anonymous* credential systems.

The list of desirable properties for an anonymous credential system is considerably larger than that for a conventional credential system. While anonymous credential systems still need to provide mechanisms that prevent users from forging credentials, they also have to provide unlinkability of transactions and anonymity of users. In certain applications, it may in addition be desirable for the system to be augmented with an 'anonymity revocation' feature. In the next chapter we present a formal model that captures five relevant properties for anonymous credential systems. While this is not an exhaustive list of the desirable properties for such systems, we believe that they are the most important ones. In the following paragraphs we give a brief literature survey of the subject.

Since the early eighties a significant amount of research has been performed on digital anonymous credential systems. Seminal papers and publications that have influenced the field are those by Chaum and his various collaborators on privacy in the digital world in

general [46, 47, 51, 52] and in the context of credentials in particular [48, 49, 50, 53, 54, 55]. As pointed out, for example, in [37, 163, 201], Chaum's proposals, as well as Damgård's proposal [63], suffer from a number of limitations, such as reliance on trusted third parties and protocols that are relatively inefficient and of limited practicality.

Subsequent work, e.g. that of Camenisch, Lysyanskaya, Persiano and others [37, 137, 163], has focussed on addressing these limitations. One of the first anonymous credential schemes to offer the levels of efficiency required by practical applications was that proposed by Chen in 1995 [56]. However, Chen's scheme still suffers from the fact that a third party is involved during the registration phase of users. Brand's proposal for a credential system, in which users can selectively disclose the attributes in their credentials [31], suffers from the fact that all issuers have to agree on a common set of security-relevant parameters. Both Brand's and Chen's schemes require users to obtain multiple copies of a credential if they intend to show it multiple times without the showings being trivially linked.

More recent proposals for constructing anonymous credential systems, such as those described in [37, 39, 137, 138, 163, 201], explicitly address a number of issues arising in previously proposed schemes. The most significant of these issues is probably that of credential non-transferability, i.e. the incorporation of mechanisms into the system that discourage users from sharing their credentials. Other requirements that are addressed by these more recent proposals are the revocation of a user's anonymity under predetermined conditions, the independence of cryptographic keys, and the support for credentials that may be shown multiple times (either up to a predetermined number, or an unlimited number).

8.1.2 Pseudonym systems

In the literature, the terms 'anonymous credential system' and 'pseudonym system' are sometimes used interchangeably. While some papers prefer the former term (e.g. [37, 39, 163]) others prefer the latter (e.g. [56, 138]). In this section we briefly discuss these two terms and the meanings assigned to them in this thesis.

An anonymous credential system essentially enables its users to interact with different organisations, i.e. the issuers and the verifiers of these credentials. In such a scheme, there is no intrinsic need for users to establish pseudonyms with the organisations. However, while

in such a case all transactions would remain unlinkable, unfortunately there would be no way for the organisations to differentiate between users.

There are situations where it is desirable for certain transactions to be unambiguously linked. In an electronic cash scheme, for example, all withdrawals of a particular user should be capable of being linked to that user's bank account. Similarly, a user of an airline miles loyalty scheme requires that his miles be deposited into his personal account whenever he shows a relevant credential.

In order to support such situations, and to provide the desired level of anonymity, users need to be able to establish a different identifier, called a *pseudonym*, with each issuer and verifier they wish to interact with. To offer privacy, it should be impossible for anyone to link together a user's pseudonyms; hence they should not possess any connection with one another or to the identity of their user (if they do, then they are clearly no longer unlinkable). In a scheme that supports such pseudonyms, users can establish different identities with different organisations. This enables users to build reputations, and organisations to hold users accountable for their actions (within the scope of a given pseudonym). At the same time it remains infeasible for colluding issuers and verifiers to decide whether or not any given pair of pseudonyms belongs to the same user; this provides a degree of privacy protection for users.

Anonymous credential systems that support such unlinkable pseudonyms are called pseudonym systems. In other words, one can view a pseudonym system as an 'enhanced' anonymous credential system; i.e. one that supports pseudonyms. Such systems obviously need to provide a mechanism for pseudonyms to be established. If, however, one wanted to use a pseudonym system as a simple anonymous credential system (i.e. without using pseudonyms as a means to differentiate between users), one could simply establish a new pseudonym for each transaction and discard it afterwards. In this sense, an anonymous credential system is a special case of a pseudonym system.

In the remainder of this thesis we are only concerned with anonymous credential systems that support pseudonyms, i.e. with pseudonym systems. In fact, we use the two terms interchangeably, as does much of the literature. In the next chapter we are not concerned with any particular pseudonym system; the interested reader is referred to the references given above. Instead, we give definitions of security and privacy that apply to such systems,

regardless of the cryptographic primitives and techniques that are used by any particular scheme.

In the remainder of this chapter, we consider practical limits to the level of pseudonym unlinkability (and, thus, subject privacy) offered by anonymous credential systems. In particular, assuming the soundness and security of such a system, we consider how timing attacks, launched by colluding issuers and verifiers, may affect pseudonym unlinkability. We also outline possible pragmatic approaches to minimising exposure to such attacks.

The remainder of this chapter is structured as follows. The next section outlines the assumptions we make about an anonymous credential system, section 8.2.1 discusses the issue of encoding freshness into credentials, and section 8.2.2 presents the timing attacks. Section 8.2.3 provides some simple heuristics to counter the attacks and section 8.3 concludes, giving directions for further research.

8.2 Timing attacks against anonymous credential systems

As discussed above, a number of anonymous credential systems have been proposed in the literature, each with its own particular set of entities, underlying problems, assumptions and properties. This section presents the model of anonymous credential systems on which the rest of this chapter is based.

We consider an anonymous credential system to involve three types of player: subjects, issuers and verifiers. We refer to issuers and verifiers, collectively, as 'organisations'. It is assumed that subjects establish at least one pseudonym with each organisation with which they wish to interact. These pseudonyms are assumed to be indistinguishable, meaning that they do not bear any connection to the identity of the subject they belong to. We further assume that pseudonyms are unlinkable, i.e. two pseudonyms for the same subject cannot be linked to each other. Subjects may obtain credentials, i.e. structures that encode a well-defined, finite set of attributes together with their values, from issuers. They may subsequently show those credentials to verifiers, i.e. to convince them that they possess (possibly a subset of) the encoded attributes. A credential is issued under a pseudonym that the subject has established with its issuer, and is shown under the pseudonym that the subject has established with the relevant verifier.

It is assumed that the anonymous credential system is sound. This means that it offers *pseudonym owner protection*, i.e. that only the subject that established a given pseudonym can show credentials under it. Soundness also implies credential *unforgeability*; the only way that subjects may prove possession of a credential is by having obtained it previously from a legitimate issuer. In some applications, it is required that the system offers the stronger property of credential *non-transferability*. This property guarantees that no subject can prove possession of a credential that it has not been issued using one of the pseudonyms that it legitimately possesses, even if the subject colludes with other subject(s) that may have (legitimately) obtained such a credential. (For formal definitions of these properties see chapter 9.) In other words, a system that offers non-transferability prohibits credential sharing, whereas a system that offers only unforgeability, does not. (Of course, the degree of protection against credential sharing is always limited, since if one subject gives all its secrets to another subject then the latter subject will always be able to impersonate the former and use its credentials.) For the purposes of the discussion in this chapter we require that credentials are bound to the subject to which they have been issued. We therefore assume that subjects do not share their credentials.

It is assumed further that the system properly protects privacy, in that a subject's transactions with organisations do not compromise the unlinkability of its pseudonyms. As pointed out in the previous chapter, this unlinkability can only be guaranteed up to a certain point, as credential *types* potentially reveal links between pseudonyms. The type of a credential is defined as the collection of attribute values that are encoded into the credential. An organisation that, for example, issues demographic credentials containing the fields `sex` and `age group`, with possible values of {`male,female`} and {`18-,18-30,30-50,50+`} respectively, may actually issue up to 8 different types of credential (one for each combination of values). To see how credential types can be exploited to link a subject's pseudonyms, consider the following trivial scenario. At time τ, a credential of type t is shown under the pseudonym p. However, suppose that, up to time τ, only one credential of type t has been issued, and this was done under pseudonym p'. It follows, under the assumption that credentials are bound to subjects, that the two pseudonyms p, p' belong to the same subject; the colluding organisations can thus successfully link those two pseudonyms.

We note that, as part of credential showing, some anonymous credential systems allow subjects to reveal only a subset of the encoded attributes; in the above example it may be possible for the subject to reveal only the value of `sex`. For these systems, it is tempting to

define the type of a credential as the collection of attributes that is revealed to the verifier during showing. However, we restrict our attention to the scenario where the verifier, rather than the subject, selects the attributes to be revealed during credential showing. This is, as far as our analysis is concerned, equivalent to the case where only the required set of attributes is encoded into a credential in the first place. This scenario is also likely to be valid for the case where verifiers perform attribute-based access control.

8.2.1 Encoding freshness into credentials

In certain applications, typically those involving short-lived credentials, verifiers need to validate the freshness of credentials[8.1]. Thus, some indication of freshness has to be encoded into a credential by its issuer. However, this indication constitutes an additional attribute, and its value helps determine the type of the credential. If the indication of freshness is unique for each credential (such as a serial number, a counter value, or a nonce), then it becomes trivial for organisations to link pseudonyms, as every credential will have its own, unique, type. It is thus desirable, from a privacy perspective, that the indication of freshness is shared among as many credentials as possible. One possible freshness indication is a timestamp generated using a universal clock, with a sufficiently coarse accuracy. We thus henceforth assume that one of the attributes that is present in all credentials is such a timestamp.

A question that arises in this context is who decides whether or not a credential has expired. If it is the issuer, it seems more appropriate for the timestamp to indicate the time of expiry. If, on the other hand, it is the verifier, then it makes more sense for the timestamp to indicate the time of issue. Since the latter alternative enables verifiers to have individual policies with respect to expiry of credentials, in the sequel we assume that the timestamp indicates the time of credential issue. We do not consider the case where two timestamps are encoded into the credential, indicating both the beginning and the end of its validity period. We further assume that all issued timestamps are given a value of $n\tau_i$ for some integer n, and some fixed time interval τ_i. The value of τ_i should be large enough to provide the degree of anonymity required. Without loss of generality we assume that all credentials issued between times $n\tau_i$ and $(n+1)\tau_i$ are given the timestamp $n\tau_i$.

[8.1] One such application is the use of anonymous credentials as user authentication tokens in an SSO context, an idea that is explored further in the following chapters of this thesis. As the timing attacks described in this chapter are a particular concern in scenarios involving short-lived credentials, it is important to be aware of them in order to be able to put into perspective the material in the following chapters.

8.2.2 Timing attacks

We consider certain attacks that may be launched by colluding organisations who wish to link pseudonyms that belong to the same subject. It is sufficient for the organisations to link the events of credential issuing and showing as corresponding to the same subject; this amounts to linking the pseudonyms that correspond to those events. We distinguish between two attack strategies. As both of them exploit temporal information in the system, they both fall into the category of timing attacks.

The first strategy does not exploit the timestamps that are encoded into the credentials. Instead, it exploits the behaviour of subjects in certain scenarios, i.e. the high likelihood that a subject will show a credential to a verifier soon after it was issued to them. That is, if a credential is issued at time τ and subsequently shown at time $\tau + \tau_\delta$, where τ_δ is small, then the issuer and verifier could collude to learn (with high probability) that the two pseudonyms involved belong to the same subject. Of course, the meaning of 'small' here will depend on the application, in particular on the rate of credential issuing. As a result, this timing attack is not equally serious in all scenarios. While in some applications (e.g. driving licences) it may not be a concern at all, in others (e.g. tickets, electronic cash, authentication tokens) the threat may be much more significant.

The second strategy is essentially the one already mentioned in section 8.2.1 above, namely the correlation of issuing and showing events based on the type of the credentials involved, and thereby linking the associated pseudonyms. However, the fact that we are now assuming that credentials encode timestamps (whose freshness is checked by verifiers), guarantees that credentials issued in different periods are of different types. Exploiting this particular fact may render the generic correlating-by-type attack much more effective. Moreover,

8.2.3 Countermeasures

An obvious countermeasure to the first attack strategy is to require subjects to wait between obtaining and showing a credential. However, this needs to be managed carefully, since simply imposing some fixed waiting time, say τ_w, does not in practice reduce the exposure to the attack. This is because correlation between issuing and showing of credentials can still be performed by pairing these events if they are separated by a time difference of a little

greater than τ_w. Thus, the delay τ_w should be randomised in some way. This, however, raises new questions: what are the minimum and maximum acceptable values for τ_w, given that it is likely (if not certain) to affect the system's usability? How does the choice of these limits affect unlinkability? How does the probability distribution according to which τ_w is chosen affect unlinkability?

The second attack scenario requires slightly different countermeasures. The objective here is to require subjects *not* to show credentials of some type until sufficiently many credentials of that type have been issued (to other subjects). Again, an obvious countermeasure is to require some (randomised) delay between the issuing and showing of credentials. The requirement by verifiers to validate the timestamps of credentials, however, introduces an additional constraint, namely that, at the time of showing, credentials should not have expired.

We now describe a simple heuristic that, using random delays, tries to approach a reasonable compromise between usability, security and privacy in the face of the above timing attacks. It requires issuers to generate credentials in batches, containing a range of consecutive issue timestamps. More precisely, suppose a subject requests an issuer to provide a credential encoding a certain set of attributes α. Instead of issuing a single credential with type defined by the concatenation of α with the current timestamp $n\tau_i$, the issuer generates a set of k credentials with types defined by the concatenation of α with the timestamps $(n-j)\tau_i$, where $0 \leq j \leq k-1$, where k is a policy-based parameter (which may be chosen by the subject or by the issuer, depending on the system context).

This means, of course, that timestamps no longer precisely encode the time of issue; it is assumed, however, that this does not affect security since those credentials with 'old' timestamps are bound to expire sooner than those with current ones. It is also required that subjects maintain a loosely synchronised clock, such that they can distinguish time periods. Further, it is assumed that verifiers accept credentials that were issued during the k most recent periods (including the current one).

When showing a credential, the subject must follow a two stage process. Note that we suppose that the set of k credentials were actually issued at time τ_{is}, where $n\tau_i \leq \tau_{is} < (n+1)\tau_i$.

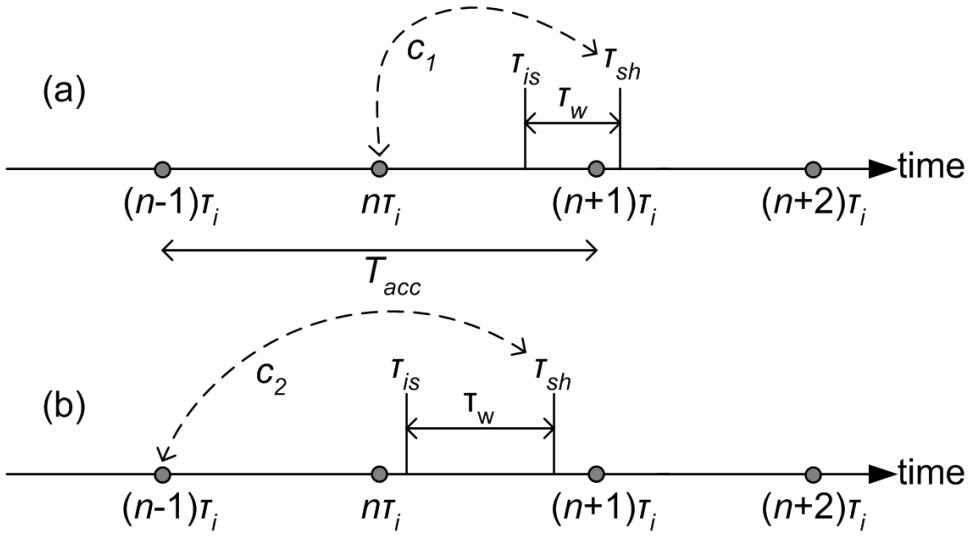

Figure 8.1: Example for $k = 2$.

1. The subject is not permitted to show any of the issued credentials until a randomised waiting time τ_w has elapsed, where $\tau_{\texttt{min}} \leq \tau_w \leq \tau_{\texttt{max}}$, and $\tau_{\texttt{min}}$ and $\tau_{\texttt{max}}$ are domain specific parameters. Clearly $\tau_{\texttt{min}}$ will depend on the rate of credential issue and showing, and should be chosen to prevent simple correlation between credential issue and showing. Further, $\tau_{\texttt{max}}$ must satisfy $\tau_{\texttt{max}} < (n+k)\tau_i - \tau_{is}$, since otherwise all the credentials may have expired before they can be used. In addition, $\tau_{\texttt{max}}$ should be large enough to sufficiently randomise the delay between issue and showing, but not so large as to damage usability. The precise choices for both parameters will be a sensitive issue, and these parameters can be subject-specific. Similarly, the probability distribution to be used to randomly select the waiting time could be varied, but it seems reasonable to use a uniform distribution.

2. The subject should then show the credential with the oldest timestamp which is still valid under the policy of the verifier. That is, if we assume that the credential is to be shown at time τ_{sh} (where $\tau_{sh} \approx \tau_{is} + \tau_w$), then the subject should show the credential with timestamp $(\lfloor \tau_{sh}/\tau_i \rfloor - k + 1)\tau_i$. Observing that $n\tau_i \leq \tau_{is}$ and that $0 \leq \tau_w < (n+k)\tau_i - \tau_{is}$, we have $n \leq \tau_{sh}/\tau_i < (n+k)$ and hence $n \leq \lfloor \tau_{sh}/\tau_i \rfloor \leq n+k-1$. Thus $(n-k+1)\tau_i \leq (\lfloor \tau_{sh}/\tau_i \rfloor - k + 1)\tau_i \leq n\tau_i$, and hence such a credential exists within the set of credentials that was issued to the subject.

Figure 8.1 illustrates the operation of the heuristic for the case $k = 2$, i.e. in the case where the verifier accepts credentials that were issued during the last two periods (this period of

time is denoted τ_{acc} in the figure). At time τ_{is} the subject obtains two credentials: c_1 and c_2, where c_1 contains timestamp $n\tau_i$ and c_2 contains timestamp $(n-1)\tau_i$. A random waiting time t_w is then selected. In case (a), the waiting time is such that τ_{sh} falls within the next period, so credential c_1 is shown (c_2 has expired). In case (b), τ_{sh} falls within the same period, so the 'older' credential c_2 is shown.

The limit $\tau_{\texttt{min}}$ should be selected such that, in every time period of that length, the expected number of subjects obtaining credentials of the type in question is sufficiently large. Imposing this lower limit is necessary because otherwise organisations may unambiguously link a pair of showing and issuing events whenever a subject selects a waiting time no more than $\tau_{\texttt{min}}$.

In some applications it may be unacceptable for a subject to obtain k credentials instead of one. A simple variation of the above heuristic does not require the subject to obtain k credentials. This involves simply choosing the waiting time τ_w first, and then only obtaining the credential that is appropriate for showing at time $\tau_{is} + \tau_w$. This variation, however, offers significantly less protection against attacks based on correlation of credential types, since the number of issued credentials drops by a factor of k.

There are trade-offs between the choices for k and τ_i. If τ_i is relatively large then k can be made smaller, saving work in credential issue and storage. However, choosing a large value for τ_i also has disadvantages; in particular it decreases the precision available for specifying lifetimes of credentials, bearing in mind that choices for k may vary across the populations of users and organisations (depending on their privacy requirements).

We note that some cryptographic constructions allow for zero-knowledge proofs (see section 1.1.5.4) of the fact that some variable lies within a certain range, without revealing its value. Anonymous credential systems that make use of such constructions (see, for example, [31]) may overcome the timing attacks introduced by timestamps, since subjects can convince verifiers that timestamps lie within some acceptable range, without revealing any information beyond this fact. In order to maximise unlinkability in this scenario, however, this range has to be selected carefully. Assuming that timestamps encode the time of issue, the lower limit of the range should indicate 'just before expiry', i.e. the subject should prove that the issue timestamp is at least $\tau_{sh} - \tau_{acc}$, where τ_{acc} denotes the time after which credentials expire (as defined by the policy of the verifier).

8.3 Concluding remarks

Timing issues often arise in the context of privacy preserving schemes, for example in the context of mix networks. Mix networks have been proposed as a solution to the problem of anonymous communication [45, 177]. They enable users to send messages to other users in a way that preserves the anonymity of the sender. A cascade of trusted parties, called 'mixes', provides the anonymising service, i.e. it hides the identities of the senders from both recipients and attackers that might analyse the traffic that is flowing through the network. Nevertheless, each mix is susceptible to timing attacks that try to correlate incoming and outgoing messages [125]. A typical countermeasure requires the mix to wait for a number of incoming messages before forwarding them (in shuffled order) to the next mix in the cascade (or, in the case of the final mix, to the recipient of the message).

Mix networks rely on third parties. Unfortunately, such entities do not exist in anonymous credential systems. In fact, in many scenarios, relying on third parties may be undesirable. In this chapter we outlined some heuristics that offer protection against timing attacks that arise when using anonymous credentials. Although they do not rely on a third party, they come with a cost in terms of communications and computational overhead and a potential impact on usability.

There is a need to devise solutions which address the threat of timing attacks on anonymous credentials. These solutions should minimise three, probably contradicting, requirements: the probability of pseudonym linking, the inconvenience introduced to subjects, and the impact on system security. In other words, they should offer reasonable tradeoffs between usability, security and privacy protection. In the next chapter we describe a system that aims to perform this job; it reduces the exposure to the attacks described in this chapter.

Chapter 9

A security model for anonymous credential systems

Contents

This chapter formally defines complexity-theoretic security and privacy notions that apply to anonymous credential systems. The relationship between these notions is explored, and it is shown that some notions imply others. Moreover, the model potentially enables reductions to be formulated that prove the security of cryptosystems that satisfy the definitions, and provides insight into the issues that arise when designing and analysing such systems. Some of the material in this chapter was previously published in [154].

9.1 Introduction

We saw in the previous chapter that anonymous credential or 'pseudonym' systems allow users to interact with organisations using distinct and unlinkable pseudonyms. In particular,

they enable a user to obtain a credential (a statement of a designated type that attests to one or more of the user's attributes) from one organisation and then 'show' it to another, such that the two organisations cannot link the issuing and showing acts; the user's transactions remain unlinkable. As was also shown in the previous chapter, this unlinkability is limited; if only one credential is ever issued with a particular set of attributes, then clearly all credential showings containing this set of attributes can be linked to each other and to the unique issued credential. So there exists an inherent limit to the unlinkability that can be obtained in such a system. In this chapter we explore this limit further.

Pseudonym systems also have to prevent users from showing credentials that have not been issued, and prevent them from pooling their credentials (for example, to collectively obtain a new credential that each user individually would not be able to). In this chapter we explore these security notions in greater detail, as well as a further security notion that we call 'pseudonym owner protection'.

When analysing the security of pseudonym systems, it is desirable to relate the difficulty of breaking the system to the difficulty of solving some well-studied problem. In order to do so it is necessary to (a) define precisely what it means to 'break' the system (or, equivalently, what it means for the system to be 'secure'), and (b) provide arguments that show how these security definitions relate to the definitions of the well-studied problem. These arguments are typically made within a complexity theoretic framework [186] and are usually referred to as 'proofs of security'. The field of study encompassing such proofs is called 'provable security', and schemes that have been analysed in this way are said to be 'provably secure'. The reader interested in this research area is referred to e.g. [12, 17, 43, 118, 173].

In this chapter we are not concerned with security proofs for pseudonym systems. We are, however, interested in the definitions that capture the key security notions, and that potentially enable such proofs to be formulated for concrete systems. The definitions occur in the context of a *model* that enables the different procedures that occur in a pseudonym system to take place in a well-defined way. This model also incorporates what is called an *adversary*. This adversary captures, again in a well-defined sense, the notion of an attacker that tries to break the system. Thus, the model is an *adversarial* one.

Our model is not the first security model that has been proposed for anonymous credential systems. In the next section we briefly discuss another such model. Section 9.1.2 provides

some clear motivation for introducing our new model.

9.1.1 Universal composability and pseudonym systems

The model used by Camenisch and Lysyanskaya [37, 137] to analyse pseudonym systems is based on a model proposed by Canetti [41] (and which was also used for the analysis of key exchange protocols [44]). This model was designed in order to make it easier to analyse cryptographic schemes that are composed of different (sub)protocols; it is called the 'universally composable framework'.

In this framework, the definition of security is based on the indistinguishability between the transcripts of protocols that occur in an 'ideal' world (where a universally trusted party guarantees security), and the 'real world' (where such a party does not exist). We next explain what this means in a little more detail. The description given here is simplified. For a full exposition the interested reader is referred to the references given above.

When analysing a particular cryptographic scheme (e.g. an anonymous credential system) in this model, one first has to define the 'ideal world'. This involves specifying the players (e.g. users and organisations) in the system and the procedures by which they interact; this specification is at a high level, i.e. it does not go into the details of the cryptographic protocols. In the ideal world, security is guaranteed by an ideal (i.e. non-existent), central, trusted party T. Players never interact directly with each other in the ideal world. They only interact with (i.e. through) T. The adversary may control a number of players in the system and may also interact with T. There exists an additional entity called the 'environment' E. It is E that chooses which procedures will be executed in the system. The environment E causes a procedure to be executed by sending a message that specifies the particular procedure to the player that should initiate it. E cannot directly send messages to those players that are controlled by the adversary. Instead, E sends the messages to the adversary, which then forwards them to the intended players. The final outcome of each procedure is reported back to E.

Next, one has to define the 'real world'. The set up is very similar to the ideal world; the main difference is that T no longer exists. Players now interact directly with each other, using the concrete cryptographic protocols of the scheme. The adversary may again control

a number of players, and may also be in control of the network itself. As in the ideal world, an environment E initiates the procedures by sending messages to the players, either directly or via the adversary.

Observe that, since T guarantees security, the ideal system is by definition secure. In fact, it serves as a specification that captures what exactly it means for the cryptographic scheme to be secure in the presence of an adversary. The cryptographic scheme, whose operation is captured by the 'real world' setting, is said to *conform* to the specification (and is therefore said to be secure), if it can be shown that no environment E can distinguish, in the presence of a simulator, whether it is interacting with the ideal or the real world. The simulator provides an interface between the environment E and the players (including the adversary) of the real world. In other words, the simulator's task is to make E believe that it operates in the ideal world whenever it operates in the real one. Demonstrating the security of a system in this model then amounts to describing the operation of a simulator that successfully performs the above job.

9.1.2 Motivation for the new model

Pseudonym systems are required to satisfy a number of security notions. Intuitively, these notions are not entirely unrelated. In the above model, however, relationships among notions are somewhat hidden by the fact that the universally trusted party T takes care of them. It is also not clear how to quantify the unlinkability obtained in the system, given the inherent limit mentioned earlier. In addition, when the model is applied in [37, 137], the adversary is not allowed to corrupt players in an adaptive fashion.

In this chapter we propose an adversarial model that enables us

- to explore the relationships between different notions, and
- to quantify, to an extent, the obtained degree of unlinkability.

While the property that the adversary gets to specify the order of events is retained, our model is stronger than the one in [37], in that the adversary can also adaptively corrupt players. Moreover, it potentially leads to a precise description of the anonymity that users may enjoy when using the system.

9.1.3 What the new model does not do

Our model abstracts away from properties that do not lie at the same level of abstraction as that at which a pseudonym system operates. It does not capture 'traditional' communications security properties, such as entity authentication or key establishment. This is not an omission; these issues are outside the scope of the model, and other models can be used to reason about these issues. Of course, if users do not authenticate organisations, and if the communications in the system are not appropriately protected at the session level, then there cannot be any security. What it means for communications to be 'appropriately protected at the session level' may vary from system to system. Typically, it means that each protocol execution is bound to exactly one conversation between the involved parties. It may also imply protection of the integrity and confidentiality of the communications occurring within the session. However, the way these services are provided lies at a different level of abstraction. We therefore assume that they are provided by the infrastructure that allows users and organisations to communicate. We also assume that, within this infrastructure, users remain anonymous to organisations (i.e. we assume an anonymous user-to-organisation channel).

The remainder of the chapter is organised as follows. The next section presents the formal model of pseudonym systems, and illustrates how the protocols of a concrete pseudonym system, namely that of Chen [56], meet the definition. Section 9.2.2 establishes the notions of pseudonym owner protection, credential unforgeability and credential non-transferability, which together capture the notions of soundness for a scheme. Further, section 9.2.3 provides a brief discussion of the notions and explains the relationships between them. It also argues, on a high level, that Chen's scheme is sound. Section 9.2.4 establishes the notion of pseudonym unlinkability. Section 9.2.5 provides a discussion of unlinkability and argues that Chen's scheme meets this definition as well. Section 9.2.6 then establishes the notion of pseudonym indistinguishability and shows it is a necessary condition for unlinkability. Finally, section 9.2.7 addresses the issue of anonymity in pseudonym systems, while section 9.3 concludes the chapter with a brief summary.

9.2 Security of pseudonym systems

In this section we describe our model of a pseudonym system. We regard a pseudonym system as being comprised of the players in the system and the procedures through which they interact. The players are divided into users, issuing organisations and verifying organisations. Since users are known to each organisation under a different pseudonym, indeed possibly under multiple pseudonyms, a procedure must be in place according to which a user and an organisation establish a new pseudonym; we call this the 'pseudonym establishment protocol'. Procedures must also be in place that allow users to obtain credentials (using the pseudonym that was established with the issuer) and to show them (using the pseudonym that was established with the verifier). We call the former the 'credential issuing protocol' and the latter the 'credential showing protocol'.

In our model, credential types are in one-to-one correspondence with (combinations of) user attributes; in other words, each combination of attributes defines a credential type. In our model, an organisation that, for example, issues demographic credentials containing the fields `sex` and `age group`, with possible values of {`male,female`} and {`18-,18-30,30-50,50+`} respectively, may actually issue credentials of up to eight different types (one for each combination of values).

9.2.1 The model

A protocol `prot` is assumed to be a tuple of interactive Turing machines [186]; an execution of `prot` is said to be successful if and only if all machines accept. The set of all non-zero polynomial functions in the natural number k is denoted by $\texttt{poly}(k)$. A real-valued function $\epsilon : \mathbb{N} \to \mathbb{R}$, is said to be *negligible* in k if and only if $0 \leq \epsilon(k) < 1/|q(k)|$ for any $q \in \texttt{poly}(k)$ and for all sufficiently large k.

Remark 1 We are concerned in this chapter with situations where two functions f and g satisfy $f(k) > g(k) + \epsilon(k)$ for any negligible function ϵ and for all sufficiently large k. To simplify the discussion we abuse our notation slightly and simply say that f is greater than $g + \epsilon(k)$, i.e. we omit explicit references to k, and we also omit the rider 'for all sufficiently large k'.

We now give a formal definition of an anonymous credential system.

Definition 1 A pseudonym system is a tuple

$$(U, I, V, k, \texttt{init}, P, T, \texttt{peprot}, \texttt{ciprot}, \texttt{csprot})$$

whose elements are as follows.

- U is the set of users. Its elements are probabilistic polynomial time Turing machines.
- I is the set of credential issuing organisations ('issuers' in short). Its elements are probabilistic polynomial time Turing machines.
- V is the set of credential verifying organisations ('verifiers' in short). Its elements are probabilistic polynomial time Turing machines.
- P is the set of pseudonyms.
- T is the set of credential types.
- k (a natural number) is the system security parameter.
- `init` is the system's probabilistic, polynomial time initialisation algorithm; on input (U, I, V, k), it outputs descriptions of the sets P and T. Depending on the particular scheme, it may also output public parameters and private values for (a subset of) the players in the scheme, i.e. users, issuers and verifiers. Furthermore, the `init` algorithm outputs a set $T_i \subseteq T$ for each $i \in I$. The set T_i represents the credential types that the issuer i will issue in the future[9.1]. It is required that, for all distinct $i, i' \in I$, $T_i \cap T_{i'} = \emptyset$. (This is easily achieved by having a unique identifier of each i embedded into all its types T_i.) The set of active credential types in the system is denoted by $T^* \stackrel{def}{=} \bigcup_{i \in I} T_i$.
- `peprot` is the *pseudonym establishment protocol*: any user/organisation pair $(u, o) \in U \times (I \cup V)$ may execute `peprot`; if the protocol succeeds, u and o will have established a pseudonym $p \in P$ and we write $\texttt{peprot}_{u,o,p}$. The probability distribution according to which p is chosen from P is defined by the protocol. (The user u is called the *owner* of p, and will typically also possess some private output associated with p as necessary to engage in `ciprot` and `csprot`.)

[9.1] In certain existing pseudonym systems, e.g. [37], credential types are identified with some form of public verification key. While these keys are typically published, their private counterparts are kept secret by the corresponding issuer.

- `ciprot` is the *credential issuing protocol*: any user/issuer pair $(u, i) \in U \times I$ may execute `ciprot` with respect to a pseudonym $p \in P$ associated with u and i (established using `peprot`) and for a credential type $t \in T_i$. If successful, we say that i has issued a credential of type t on pseudonym p, and we write $\texttt{ciprot}_{i,p,t}$.
- `csprot` is the *credential showing protocol*: any user/verifier pair $(u, v) \in U \times V$ may execute `csprot` with respect to a pseudonym $p \in P$ associated with u and v (established using `peprot`) and for a particular credential type $t \in T$; if the protocol succeeds we say that u has shown a credential of type t on pseudonym p to v, and we write $\texttt{csprot}_{v,p,t}$.

Note that, since `init` runs in polynomial time, $|U|, |I|, |V|, |T^*| \in \texttt{poly}(k)$. We can now define what it means for a pseudonym system to be correct.

Definition 2 A pseudonym system is said to be *correct* if and only if executions of `peprot`, `ciprot` and `csprot` are successful with probability greater than $1 - \epsilon(k)$ for some negligible function ϵ.

At this point it is appropriate to introduce an example pseudonym system and explain how its protocols work. The example we describe here is the pseudonym system by Chen [56]. Note that we give only simplified high-level description. For a detailed description of the protocols the reader is referred to [56]. In Chen's system, the pseudonym establishment protocol $\texttt{peprot}_{\texttt{u,o,p}}$ is such that the user's u pseudonym p appears as a random number to the organisation o. The protocol, shown in Figure 9.1, which involves a trusted party, forces the user to compute p using a factor c that is unique to u and common to all his pseudonyms. First the trusted party sends c to u (step 1). Then u forms p as a function of c and a value s that the user selects randomly and keeps secret. This computation is such that, given a pseudonym p (and without knowledge of s), it is infeasible to learn anything about s. The user then obtains from the trusted party a signature on a blinded version of p (step 2). This is accomplished using a blind signature scheme (see section 1.1.5.3). The signature attests to the fact that p is well-formed, i.e. that it is a function of c. In order to unblind this signature such that it becomes valid on p, the user must know the corresponding s. Finally, the user sends p and the trusted party's signature on p to the organisation o (step 3). The organisation verifies the signature and, if successful, registers p as a valid pseudonym.

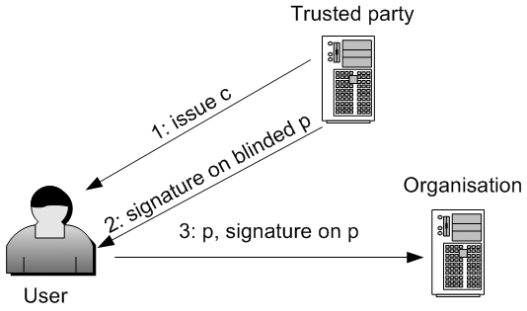

Figure 9.1: Pseudonym establishment protocol of Chen's system

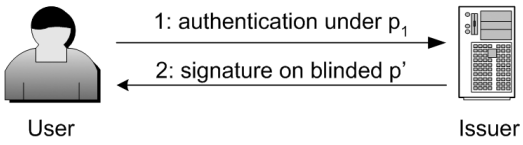

Figure 9.2: Credential issuing protocol of Chen's system

The credential issuing protocol $\texttt{ciprot}_{i,p_1,t}$ of Chen's system, shown in Figure 9.2, operates as follows. First, the user authenticates itself to the issuer i under p_1. The authentication method is not specified in Chen's system. What matters is that i is convinced that the credential of type t will be issued to an authorised user. If authentication succeeds, i issues a signature on a blinded version of p' to the user, using a private key that corresponds to t (see section 1.1.5.3). The pseudonym p' is either the user's pseudonym p_2 under which he wishes to later show the credential, or an 'intermediate' pseudonym that is not established in the system. In both cases, the issuer's blind signature has the property that it enables the user to compute a valid signature on p_2 only if he knows the secrets s_1 and s_2 that are associated with p_1 and p_2.

The credential showing protocol $\texttt{csprot}_{v,p_2,t}$ of Chen's system, shown in Figure 9.3 operates as follows. The user shows p_2 and a signature on p_2 that verifies with i's public key that corresponds to t. The user can obtain such a signature as a result of a $\texttt{ciprot}_{\texttt{i,p}_1\texttt{,t}}$ protocol as described above. The verifier checks the signature and, if valid, accepts. It is worth stressing that, since issuing a credential amounts to the issuer generating a *blind* signature, the information that the user reveals to the verifier during credential showing does not reveal any link to the information that the issuer saw during credential issuing.

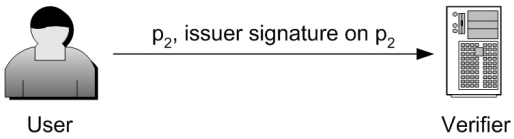

Figure 9.3: Credential showing protocol of Chen's system

9.2.2 The games and soundness

Our notions of security follow well-established techniques from the field of provable security (see, for example, [12, 17, 43, 118, 173]). In particular, each notion is defined by means of a game between two Turing machines: a Challenger and an Adversary.

In this section we define Game 1, which captures 'pseudonym owner protection', Game 2, which captures 'credential unforgeability', and Game 3, which captures 'credential non-transferability'. In sections 9.2.4 and 9.2.6 below we define Game 4 and Game 5, which capture 'unlinkability' and 'indistinguishability' of pseudonyms, respectively.

At the beginning of all games, the Challenger sets up the system by selecting the set of users U, issuers I, verifiers V, and a security parameter k and by running `init` with these as input. At this point, the Challenger controls all the players in the system. The Adversary, which is assumed to be a probabilistic polynomial time and space algorithm and is denoted by $\mathcal{A}$, then receives as input the sets U, I, V and T_i, descriptions of the sets P and T, and the system's public information.

Each of the games consists of two distinct and successive phases. During the first phase of each game, $\mathcal{A}$ may issue (oracle type) queries to the Challenger. When receiving a query, the Challenger performs a certain task and, depending on its outcome, responds in a certain way. The response is given to $\mathcal{A}$. During the second phase of the game $\mathcal{A}$ is no longer allowed to issue queries.

We now describe the six query types that $\mathcal{A}$ may issue during the first phase of Games 1, 2 and 3. The query types for Games 4 and 5 are described in sections 9.2.4 and 9.2.6 below.

`runpeprot`(u, o): $\mathcal{A}$ arbitrarily selects a user/organisation pair $(u, o) \in U \times (I \cup V)$ and issues this query. When this happens, the Challenger makes u and o execute $\texttt{peprot}_{u,o,p}$. The Challenger replies `true` if the protocol execution is successful and `false` otherwise. (If the

execution is successful, u and o will have established a new pseudonym $p \in P$; $\mathcal{A}$, however, does not learn its value.)

`runciprot`(u, i, t)**:** $\mathcal{A}$ arbitrarily selects a user/issuer/credential type tuple $(u, i, t) \in U \times I \times T_i$ and issues this query. When this happens, the Challenger selects a pseudonym p from set of pseudonyms that u and i have established and makes u and i execute $\texttt{ciprot}_{i,p,t}$. We do not specify the probability distribution according to which the Challenger selects p from the set of pseudonyms that is established between u and i, since the security notions that are defined by means of the games in which queries of this type may occur, should hold for all such distributions. The Challenger replies `true` if the protocol execution is successful and `false` otherwise (including the case where u and i have not established any pseudonym). Note that $\mathcal{A}$ does not learn the value of p.

`runcsprot`(u, v, t)**:** $\mathcal{A}$ arbitrarily selects a user/verifier/credential type triple $(u, v, t) \in U \times V \times T$ and issues this query. When this happens, the Challenger selects a pseudonym p from the set of pseudonyms that u and v have established, and makes u and v execute $\texttt{csprot}_{v,p,t}$. He replies `true` if the protocol execution is successful and `false` otherwise (including the case where u and v have not established any pseudonym). Note that $\mathcal{A}$ does not learn the value of p.

`corruptUser`(u)**:** $\mathcal{A}$ arbitrarily selects a user $u \in U$ and issues this query. When this happens, the Challenger hands all the private information of u to $\mathcal{A}$. This includes the pseudonyms, credentials and all the past protocol views of u. From that point on, the control of u is passed from the Challenger to $\mathcal{A}$.

`corruptIssuer`(i)**:** $\mathcal{A}$ arbitrarily selects an issuer $i \in I$ and issues this query. When this happens, the Challenger hands all the private information of i to $\mathcal{A}$. This includes the set of pseudonyms i has established and all its past protocol views. From that point on, the control of i is passed from the Challenger to $\mathcal{A}$.

`corruptVerifier`(v)**:** $\mathcal{A}$ arbitrarily selects a verifier $v \in V$ and issues this query. When this happens, the Challenger hands all the private information of v to $\mathcal{A}$. This includes the set of pseudonyms v has established and all its past protocol views. From that point on, the control of v is passed from the Challenger to $\mathcal{A}$.

In all games, a global and monotonically increasing variable τ counts $\mathcal{A}$'s queries. We say that the query is issued at the time indicated by τ. At a certain point in time, $\mathcal{A}$ exits the first phase and enters the second phase. The value of τ at that point is denoted by $\tau_{\texttt{max}}$. In the second phase $\mathcal{A}$ may no longer issue any queries; what takes place is specific to each game and is described below.

To describe the games we require some additional notation. In the following, $P_{u,o} \subseteq P$ denotes the set of pseudonyms the user $u \in U$ has established with the organisation $o \in (I \cup V)$ at time $\tau_{\texttt{max}}$ (via $\mathcal{A}$'s `runpeprot` queries), i.e.

$$P_{u,o} \stackrel{def}{=} \{p \in P \mid \text{a successful } \texttt{peprot}_{u,o,p} \text{ occurred at a time } \tau \leq \tau_{\texttt{max}}\}.$$

The set of pseudonyms belonging to u is defined as $P_u \stackrel{def}{=} \bigcup_{o \in (I \cup V)} P_{u,o}$ and the set of pseudonyms that o has established is defined as $P_o \stackrel{def}{=} \bigcup_{u \in U} P_{u,o}$. (Since $\mathcal{A}$ does not learn the value of pseudonyms during their establishment, only u knows P_u and only o knows P_o.) The set of active pseudonyms in the system is defined as $P^* \stackrel{def}{=} \bigcup_{u \in U} P_u$, or, equivalently, $P^* \stackrel{def}{=} \bigcup_{o \in (I \cup V)} P_o$. Since $\mathcal{A}$ is polynomially bounded in k, it holds that $|P^*| \in \texttt{poly}(k)$. It is required that, for all distinct $u, u' \in U$, $P_u \cap P_{u'} = \emptyset$. This requirement is a technicality that we need in order to define the function f. It practice it can be met by having `peprot` select pseudonyms uniformly at random from a large enough set P. The pseudonym establishment protocols of some existing schemes are of this form. The function $f : P^* \to U$ maps pseudonyms to their owners, which is well-defined by the assumption that $P_u \cap P_{u'} = \emptyset$.

Let $\hat{U} \subseteq U$, $\hat{I} \subseteq I$ and $\hat{V} \subseteq V$ denote the subsets of users, issuers and verifiers respectively that $\mathcal{A}$ corrupted during the first phase. Further, let $P_{u,t}(x) \subseteq P_u$ denote the subset of pseudonyms belonging to user $u \in U$ on which a credential of type $t \in T^*$ has been issued prior to time x, i.e. $P_{u,t}(x) \stackrel{def}{=} \{p \in P_u \mid \text{a successful } \texttt{ciprot}_{p,\cdot,t} \text{ occurred at time } \tau \leq x\}$.

We now describe the second phase of Games 1, 2 and 3. As mentioned above, $\mathcal{A}$ may no longer issue queries to the Challenger in this phase. He may, however, engage in $\texttt{ciprot}_{i,p,t}$ and $\texttt{csprot}_{v,p,t}$ executions directly with organisations (while pretending to be the user $f(p)$).

Game 1 (pseudonym owner protection): $\mathcal{A}$ selects a pseudonym/verifier/type triple $(p, v, t) \in P^* \times (V - \hat{V}) \times T$ such that $f(p) \in (U - \hat{U})$. We say that $\mathcal{A}$ wins the game iff he can make v accept in a $\texttt{csprot}_{v,p,t}$ execution.

Game 2 (credential unforgeability): $\mathcal{A}$ selects a pseudonym/verifier/type triple $(p, v, t) \in P^* \times (V - \hat{V}) \times (T - \bigcup_{i \in \hat{I}} T_i)$ such that $P_{f(p),t}(\tau_{\texttt{max}}) = \emptyset$ and $\bigcup_{u \in \hat{U}} P_{u,t}(\tau_{\texttt{max}}) = \emptyset$. We say that $\mathcal{A}$ wins the game iff he can make v accept in a $\texttt{csprot}_{v,p,t}$ execution.

Game 3 (credential non-transferability): $\mathcal{A}$ selects a pseudonym/verifier/type triple $(p, v, t) \in P^* \times (V - \hat{V}) \times (T - \bigcup_{i \in \hat{I}} T_i)$ such that $P_{f(p),t}(\tau_{\texttt{max}}) = \emptyset$. We say that $\mathcal{A}$ wins the game iff he can make v accept in a $\texttt{csprot}_{v,p,t}$ execution.

Definition 3 A pseudonym system is said to offer pseudonym owner protection, credential unforgeability or credential non-transferability if and only if no adversary $\mathcal{A}$ can win Game 1, 2 or 3, respectively, with a probability greater than any negligible function in k.

9.2.3 Discussion

Game 1, 'pseudonym owner protection', captures the property that nobody — even when colluding with users, issuers and verifiers — should be able to successfully show a credential on a pseudonym of which he is not the owner (i.e. on a pseudonym which was not established by himself). The property is typically achieved by having the pseudonym establishment protocol generate certain private output for the user. This output is then treated as a secret that enables the user to authenticate himself as the pseudonym owner during the execution of the credential issuing and showing protocols.

In Chen's system, the user's private output from $\texttt{peprot}_{\texttt{u,o,p}}$ is the secret value s that used to form the pseudonym p. This secret is not revealed by any of the sytem's protocols and knowledge of it is necessary in order to show a credential under p.

Games 2 and 3 capture security for organisations and users. In particular, Game 2 captures what is usually perceived as 'credential unforgeability'. If a (dishonest) user can construct a credential by himself (i.e. without obtaining it legitimately from an issuing organisation), if, in other words, the user can *forge* the credential, then the system clearly does not offer credential unforgeability. Game 2 captures unforgeability in this sense. There is, however, a simplistic way for a user to 'forge' a credential, namely by 'borrowing' it from another user with whom he colludes (and who legitimately obtained the credential from an issuing organisation). This type of 'forgery' is not captured by Game 2. In some applications credential sharing is not a concern, while forgery is.

In Chen's system, unforgeability of credentials is guaranteed by the blind signature scheme employed and the way pseudonyms are constructed. Section 4 of [56] provides a detailed discussion of unforgeability.

Game 3, credential non-transferability, captures the case of credential sharing between users. In a system that satisfies the credential non-transferability property, no user can successfully show, using one of his own pseudonyms, a credential of a type he was never issued. This holds even if the user colludes with other users that have been issued credentials of that type. Note that the case where the user attempts to show a credential using *someone else's* pseudonym (with whom he colludes and who might have legitimately obtained a credential) is not captured by the game. We call this latter type of credential transfer 'pseudonym sharing'. In fact, if users share all their secrets, it is impossible to prevent pseudonym sharing, and thereby credential sharing.

In Chen's pseudonym system, non-transferability is guaranteed by the fact that all pseudonyms are a function of a unique common factor c. That is, the protocols of the system are such that it is infeasible to show a credential on a pseudonym that is a function of a different c than the pseudonym on which the credential was issued.

It is interesting to observe the relationship between the notions of unforgeability and non-transferability: the latter, being stronger, implies the former. Clearly, if a dishonest user can construct credentials by himself, then there is no need for him to collude with other users in order to forge one. In the model, this is simply reflected by the fact that the adversary is more restricted in his choice of the credential type in (the second phase of) Game 2 than he is in (the second phase of) Game 3. A system that offers non-transferability also offers unforgeability.

This relationship between unforgeability and non-transferability motivates the following definition of a sound pseudonym system.

Definition 4 A pseudonym system is said to be sound if and only if it is correct, it offers pseudonym owner protection, and it offers credential non-transferability.

Observe that it is challenging to construct pseudonym systems that prevent pseudonym sharing. Certainly, as mentioned above, if two users share all their secrets, then they can

act as each other in all circumstances. Thus, in order to prevent pseudonym sharing, one will always have to assume that users will not share *all* their secrets, either because they will be prevented by some means, e.g. by the use of tamper-resistant hardware, or because they will be given a sufficiently strong incentive not to do so. Examples of schemes that follow the latter strategy include those of Lysyanskaya [138], where sharing credentials implies sharing a highly valued key (this is called 'PKI-assured non-transferability'), and Camenisch and Lysyanskaya [37], where sharing one credential implies sharing all credentials (this is called 'all-or-nothing non-transferability').

9.2.4 Unlinkability of pseudonyms

We now define Game 4 in order to capture the first privacy property required of pseudonym systems, i.e. the property of pseudonym unlinkability. A second (weaker) privacy property is defined in section 9.2.6.

In the first phase of the Game 4, $\mathcal{A}$ is allowed to issue queries from the following set of six query types, the first three of which are similar but not identical to the first three query types of section 9.2.2.

`runpeprot`(o, Δ): $\mathcal{A}$ arbitrarily selects an organisation $o \in (I \cup V)$ and a probability distribution Δ over U and issues this query. When this happens, the Challenger selects a user u according to Δ from U and makes u and o execute $\texttt{peprot}_{u,o,p}$. The challenger replies `true` if the protocol execution is successful and `false` otherwise. (If the execution is successful, $\mathcal{A}$ knows that o has established a new pseudonym $p \in P$ with some user, but neither learns p nor the identity of its owner.)

`runciprot`(p, i, t): $\mathcal{A}$ arbitrarily selects a pseudonym/issuer/credential type triple $(p, i, t) \in P \times I \times T_i$ and issues this query. When this happens, the Challenger selects the owner of p and makes him execute $\texttt{ciprot}_{i,p,t}$ with i. He replies `true` if the protocol execution is successful and `false` otherwise (including the case where p has no owner). Note that $\mathcal{A}$ does not learn who the owner of p is.

`runcsprot`(p, v, t): $\mathcal{A}$ arbitrarily selects a pseudonym/verifier/credential type triple $(p, v, t) \in P \times V \times T$ and issues this query. When this happens, the Challenger selects the owner of p and makes him execute $\texttt{csprot}_{v,p,t}$ with v. He replies `true` if the protocol execution is

successful and `false` otherwise (including the case where p has no owner). Note that $\mathcal{A}$ does not learn who the owner of p is.

`corruptUser`(u)**:** As in section 9.2.2.

`corruptIssuer`(i)**:** As in section 9.2.2.

`corruptVerifier`(v)**:** As in section 9.2.2.

We now describe the second phase of Game 4. We denote the set of pseudonyms that belong to uncorrupted users by $P^{**} \stackrel{def}{=} P^* - \bigcup_{u \in \hat{U}} P_u$.

Game 4 (pseudonym unlinkability): $\mathcal{A}$ outputs two distinct pseudonyms $p_1, p_2 \in P^{**}$. We say that $\mathcal{A}$ wins the game if and only if $f(p_1) = f(p_2)$.

$\mathcal{A}$ may apply a variety of strategies in his effort to correlate pseudonyms. We now consider what is perhaps the most naive strategy and arrive at the following simple result.

Lemma 1 If, during all `runpeprot`(o, Δ) queries in an instance of Game 4, Δ is the uniform distribution (i.e. users are selected uniformly at random), and two pseudonyms, p_1, p_2 say, are chosen at random from P^{**}, then the probability that $f(p_1) = f(p_2)$ is $1/|U - \hat{U}|$.

Proof Suppose $f(p_1) = u \in (U - \hat{U})$. Then the probability that $f(p_2) = u$ is $1/|U - \hat{U}|$, since all users are allocated uniformly at random to pseudonyms. The result follows. □

Thus it is tempting to define a pseudonym system that offers unlinkability of pseudonyms as a system where $\mathcal{A}$ cannot win Game 4 with probability greater than $1/|U - \hat{U}| + \epsilon(k)$ for any negligible function ϵ. However, this is only a reasonable definition of unlinkability if the following two conditions hold.

1. In all `runpeprot`$(\cdot, \Delta)$ queries, Δ is the uniform distribution.
2. No credentials are shown during the first phase of the game, i.e. there are no instances of `runcsprot`.

We now examine the situation when these conditions do not hold. Suppose only the first of

the two conditions does not hold, i.e. that the distributions $\Delta_1, \Delta_2, \ldots, \Delta_{|P^*|}$ according to which users are assigned to the pseudonyms $p_1, p_2, \ldots, p_{|P^*|} \in P^*$, are arbitrary probability distributions over U. We arrive at the following generalisation of Lemma 1.

Lemma 2 If, in the second phase of Game 4, two pseudonyms, p_i and p_j say, are chosen from P^*, then the probability that $f(p_i) = f(p_j)$ is equal to

$$P_{p_i,p_j} = \sum_{u \in U} P_{\Delta_i}(u) P_{\Delta_j}(u) \tag{9.1}$$

where Δ_i and Δ_j are the probability distributions that were used in the queries in which p_i and p_j were established, and $P_{\Delta_i}(u)$ denotes the probability that user u was selected during the `runpeprot`$(\cdot, \Delta)$ query that resulted in the establishment of the pseudonym p_i.

Proof For any given $u \in U$, the probability that $f(p_i) = u$ is, by definition, $P_{\Delta_i}(u)$, and the probability that $f(p_j) = u$ is $P_{\Delta_j}(u)$. The result now follows. □

By Lemma 2, in $\mathcal{A}$'s view, the probability that $f(p_i) = f(p_j)$ is given by equation 9.1. The number of distinct pseudonym pairs in the system is $|P^*| \times (|P^*| - 1)$, which is polynomial in k. Therefore $\mathcal{A}$ can evaluate P_{p_i,p_j} for all such pairs. It is now tempting to define a pseudonym system that offers unlinkability of pseudonyms as a system where $\mathcal{A}$ cannot win Game 4 with probability greater than

$$\max_{\substack{p_i,p_j \in P^* \\ p_i \neq p_j}} (P_{p_i,p_j}) + \epsilon(k)$$

for any negligible function ϵ. However, this still assumes that the second of the above two conditions holds.

We now examine the situation when neither of the two conditions hold. In this case, any instance of `runcsprot` potentially provides $\mathcal{A}$ with information about possible links between pseudonyms, and hence potentially increases its probability of success in linking pseudonyms. Thus, the definition of pseudonym unlinkability needs to take this additional information into account.

Assuming a sound pseudonym system, there are two types of deduction that can be made.

- Suppose a `runcsprot` invocation, say `runcsprot`(p, v, t) for some p, v and t, issued at time τ, returns `true`. Then, as a result of the non-transferability of credentials, $\mathcal{A}$ can deduce, with probability greater than $1 - \epsilon(k)$ for some negligible function ϵ, that there exists some $p' \in \bigcup_{u \in U} P_{u,t}(\tau)$ such that $f(p) = f(p')$.
- Suppose a `runcsprot` invocation, say `runcsprot`(p, v, t) for some p, v and t, issued at time τ, returns `false`. Then, as a result of the non-transferability of credentials, $\mathcal{A}$ can deduce, with probability greater than $1 - \epsilon(k)$ for some negligible function ϵ, that $f(p) \neq f(p')$ for all $p' \in \bigcup_{u \in U} P_{u,t}(\tau)$.

In any instance of Game 4, which in its first phase will involve a series of `runcsprot` queries, $\mathcal{A}$ will be able to make a series of deductions about matchings of pseudonyms based on the outcomes ({`true`,`false`}) of `runcsprot` queries, as above. Based on these observations, $\mathcal{A}$ can refine the probability P_{p_1,p_2} for each pair of distinct pseudonyms p_1, $p_2 \in P^{**}$.

We now define $\bar{P}$ to be the maximum of these probabilities, i.e.

$$\bar{P} \stackrel{def}{=} \max_{\substack{p_1,p_2 \in P^{**} \\ p_1 \neq p_2}} (P_{p_1,p_2}).$$

We can now define the notion of pseudonym unlinkability.

Definition 5 A sound pseudonym system is said to offer pseudonym unlinkability if and only if no $\mathcal{A}$ can win Game 4 with probability greater than $\bar{P} + \epsilon(k)$ for any negligible function ϵ.

In Chen's pseudonym system, unlinkability is guaranteed by that fact that (a) pseudonyms appear like random numbers to organisations, (b) issuing a credential amounts to the issuer generating a blind signature (see section 1.1.5.3) on one of the user's pseudonyms, and (c) the user can, based on such a signature, compute a valid signature on any of his other pseudonyms in a way that does not reveal any connection to the pseudonym under which the credential was issued.

Two examples of how the two types of deduction mentioned above might be applied to compute $\bar{P}$ are given in the next section.

Example Scenarios

The following two example scenarios illustrate how the adversarial strategies are captured by the probability bound $\bar{P}$.

For the sake of simplicity, in the first example there are: one issuer which issues only one type of credential, one verifier, and two users. It is assumed that, during the first phase of Game 4 (unlinkability), the adversary corrupts all parties except for the two users, i.e. $I = \hat{I} = \{i\}$, $V = \hat{V} = \{v\}$, $U = \{u_1, u_2\}$, $\hat{U} = \emptyset$ and $T^* = T_i = \{t_1\}$.

Table 9.1: Example scenario 1.

Time	*Query type*	*Parameters*	*Pseudonym*	*Type*	*Outcome*
1	`runpeprot`	$i, \{2/5, 3/5\}$	p_1	n/a	`true`
2	`runpeprot`	$i, \{4/7, 3/7\}$	p_2	n/a	`true`
3	`runpeprot`	$v, \{1/2, 1/2\}$	p_3	n/a	`true`
4	`runciprot`	i	p_2	t_1	`true`
5	`runcsprot`	v	p_3	t_1	`false`

At time $\tau = 3$ (i.e. between the third and the fourth query), $\mathcal{A}$ can calculate P_{p_i,p_j} for all pseudonym pairs, based on the probability distributions in the `runpeprot` queries and according to Equation 9.1. This yields the values $P_{p_1,p_2} = 17/35$, $P_{p_1,p_3} = 1/2$, and $P_{p_2,p_3} = 1/2$.

However, after the `runcsprot` query, $\mathcal{A}$ can deduce that $f(p_3) \neq f(p_2)$, i.e. that $P_{p_2,p_3} = 0$. This means that, either $f(p_2) = u_1$ and $f(p_3) = u_2$, or vice versa, i.e. $f(p_2) = u_2$ and $f(p_3) = u_1$.

$\mathcal{A}$ has to examine both cases: in the former, $P_{p_1,p_2} = 2/5$ and $P_{p_1,p_3} = 3/5$, and in the latter $P_{p_1,p_2} = 3/5$ and $P_{p_1,p_3} = 2/5$. Since each of two cases are equally likely, $\mathcal{A}$ can deduce that, overall, $P_{p_1,p_2} = 3/5 \times 1/2 + 2/5 \times 1/2 = 50\%$ and $P_{p_1,p_3} = 2/5 \times 1/2 + 3/5 \times 1/2 = 50\%$. Thus, if $\mathcal{A}$ outputs either (p_1, p_2) or (p_1, p_3) at the end of the game, it has a 50% winning chance. If a (sound) pseudonym system offers pseudonym unlinkability, then no $\mathcal{A}$ should be able to break this bound by a non-negligible quantity.

In the second example there are: one issuer which issues only one type of credential, one verifier, and three users. It is assumed that, during the first phase of Game 4 (unlinkability),

the adversary corrupts all parties except for the three users, i.e. $I = \hat{I} = \{i\}$, $V = \hat{V} = \{v\}$, $U = \{u_1, u_2, u_3\}$, $\hat{U} = \emptyset$ and $T^* = T_i = \{t_1\}$.

Table 9.2 depicts the queries that $\mathcal{A}$ issues in this example scenario.

Table 9.2: Example scenario 2.

Time	*Query type*	*Parameters*	*Pseudonym*	*Type*	*Outcome*
1	`runpeprot`	$i, \{1/3, 1/3, 1/3\}$	p_1	n/a	`true`
2	`runpeprot`	$i, \{1/3, 1/3, 1/3\}$	p_2	n/a	`true`
3	`runpeprot`	$i, \{1/3, 1/3, 1/3\}$	p_3	n/a	`true`
4	`runpeprot`	$v, \{1/3, 1/3, 1/3\}$	p_4	n/a	`true`
5	`runpeprot`	$v, \{1/3, 1/3, 1/3\}$	p_5	n/a	`true`
6	`runpeprot`	$v, \{1/3, 1/3, 1/3\}$	p_6	n/a	`true`
7	`runciprot`	i	p_1	t_1	`true`
8	`runciprot`	i	p_2	t_1	`true`
9	`runciprot`	i	p_3	t_1	`true`
10	`runcsprot`	v	p_4	t_1	`true`
11	`runcsprot`	v	p_5	t_1	`false`
12	`runcsprot`	v	p_6	t_1	`false`

In this example, $\mathcal{A}$ selected the uniform distribution during the establishment of all pseudonyms in queries 1–6. From the first `runcsprot` query, $\mathcal{A}$ can deduce that at least one of $f(p_4) = f(p_1)$, $f(p_4) = f(p_2)$, and $f(p_4) = f(p_3)$ must be true. From the second `runcsprot` query, $\mathcal{A}$ can deduce that $f(p_5) \neq f(p_1)$ and $f(p_5) \neq f(p_2)$ and $f(p_5) \neq f(p_3)$. From the third `runcsprot` query, $\mathcal{A}$ can deduce that $f(p_6) \neq f(p_1)$, $f(p_6) \neq f(p_2)$, and $f(p_6) \neq f(p_3)$.

Combining the three `runcsprot` queries, $\mathcal{A}$ can deduce, with certainty, that $f(p_4) \neq f(p_5)$ and that $f(p_4) \neq f(p_6)$. It follows that p_5 and p_6 must belong to the set $\{u_1, u_2, u_3\} - \{f(p_4)\}$. So, the probability P_{p_5,p_6} that $f(p_5) = f(p_6)$ is 1/2. This happens to be the maximum over all distinct pseudonym pairs and thus, in the example, $\bar{P} = 1/2$. In other words, if $\mathcal{A}$, at the end of the game, outputs (p_5, p_6), he has a 50% chance of winning the game. If a (sound) pseudonym system offers pseudonym unlinkability, then no $\mathcal{A}$ should be able to break this bound by a non-negligible quantity.

9.2.5 Discussion

In the real world, colluding organisations could come up with many strategies that can be used in order to correlate pseudonyms potentially more effectively than those discussed above. Examples include attacks that take into account information such as the time or the geographical location of events that occur in the system. These attacks, however, are not captured by the model, simply because they lie at a different level of abstraction. Protection against, say, timing attacks, de-anonymising traffic analysis or social engineering, is required irrespective of which particular pseudonym system is being used. The only adversarial strategies to correlate pseudonyms that are inherent in the system, and therefore lie at the same level of abstraction, are the following.

1. If some user is asked for, but fails to produce, a credential of a given type, the colluding organisations know that none of the pseudonyms on which a credential of that type was previously issued belongs to that user.
2. If some user successfully shows a credential of a given type on one of his pseudonyms, the colluding organisations know that at least one of the pseudonyms on which a credential of that type was previously issued belongs to that user.

These strategies are captured by the probability bound $\bar{P}$. A pseudonym system cannot protect against these strategies without breaching one of its essential properties: that of credential non-transferability. In other words, if a (sound) pseudonym system satisfies Definition 5, this means then that the probability that pseudonyms can be successfully linked does not exceed the given bound (by a non-negligible quantity), provided that no 'out-of-scope' attacks place.

9.2.6 Indistinguishability of pseudonyms

We now establish our second privacy property, namely the notion of indistinguishability of pseudonyms and show that it is a necessary condition for pseudonym unlinkability.

Consider the following game between a Challenger and a polynomial time (and space) adversary $\mathcal{A}$. First, the Challenger chooses sets of users U, issuers I, and verifiers V, a sound

pseudonym system, and a security parameter k. On input U, I, V, k, he runs `init` and gives the set U of users to $\mathcal{A}$. $\mathcal{A}$ then chooses two users $u_0, u_1 \in U$ and gives them to the Challenger. The Challenger now chooses an unbiased random bit $b \in \{0, 1\}$ and makes u_b execute $\texttt{peprot}_{u,o,p}$ with some organisation $o \in (I \cup V)$. He then gives o's private information (including the protocol view and the resulting pseudonym p) to $\mathcal{A}$.

Game 5 (pseudonym indistinguishability): $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$. We say that $\mathcal{A}$ wins the game if and only if $b' = b$.

Definition 6 A pseudonym system is said to offer indistinguishability of pseudonyms if and only if no adversary $\mathcal{A}$ can win the above game with probability $\mathbf{Pr} > 1/2 + \epsilon(k)$, for any negligible function ϵ.

We can now give our main result.

Theorem 1 If a sound pseudonym system offers pseudonym unlinkability then it also offers pseudonym indistinguishability.

Proof Suppose the converse, i.e. suppose the pseudonym system offers pseudonym unlinkability but does not offer pseudonym indistinguishability. Given $\mathcal{A}^{\mathtt{i}}$, an adversary that breaks pseudonym indistinguishability, we construct $\mathcal{A}^{\mathtt{u}}$, an adversary that breaks pseudonym unlinkability, as follows. While playing Game 4 (unlinkability) with the Challenger, $\mathcal{A}^{\mathtt{u}}$ plays the role of the Challenger in Game 5 (indistinguishability) with $\mathcal{A}^{\mathtt{i}}$.

Choose a negligible function ϵ. Let $\mu(k) = \sqrt{\epsilon(k)/2}$, which, by definition, is also negligible. In Game 4, $\mathcal{A}^{\mathtt{u}}$ corrupts all but two users, say u_0 and u_1, and one organisation, say o, i.e. $(U - \hat{U}) = \{u_0, u_1\}$ and $\hat{I} \cup \hat{V} = \{o\}$. Then $\mathcal{A}^{\mathtt{u}}$ issues $\texttt{runpeprot}(o, \Delta)$ queries until three pseudonyms, say p_1, p_2 and p_3, are established between o and $\{u_0, u_1\}$. $\mathcal{A}^{\mathtt{u}}$ chooses Δ to be the uniform distribution over U in all these queries and does not issue any other queries. Thus, both conditions listed in section 9.2.4 hold and, since $|U - \hat{U}| = 2$, $\mathcal{A}^{\mathtt{u}}$ is said to break unlinkability if and only if it can win the game with probability greater than $1/2 + \epsilon(k)$.

$\mathcal{A}^{\mathtt{u}}$ then plays three instances of Game 5 (indistinguishability) with $\mathcal{A}^{\mathtt{i}}$; in all these games he defines the set of users to be $U = \{u_0, u_1\}$ and the collection of organisations to be $I \cup V = \{o\}$. $\mathcal{A}^{\mathtt{u}}$ will use $\mathcal{A}^{\mathtt{i}}$'s ability to win instances of Game 5 in order to win, with non-negligible advantage, the instance of Game 4. To this end, in the first instance of Game 5,

he gives the pseudonym p_1 together with o's private information and corresponding `peprot` view to $\mathcal{A}^{\mathtt{i}}$. Similarly, in the second and third instances he gives p_2 and p_3 respectively (together with o's private information and corresponding `peprot` views) to $\mathcal{A}^{\mathtt{i}}$. Denote $\mathcal{A}^{\mathtt{i}}$'s output in the three instances of Game 5 by b_1, b_2 and b_3 respectively. Now, since we have assumed that $\mathcal{A}^{\mathtt{i}}$ breaks pseudonym indistinguishability, we suppose that $\mathcal{A}^{\mathtt{i}}$ wins all instances of Game 5 with probability $1/2 + \delta(k)$, where $\delta(k) > \mu(k)$ for all sufficiently large k.

$\mathcal{A}^{\mathtt{u}}$ then selects $j, j' \in \{1, 2, 3\}$, $j \neq j'$, such that $b_j = b_{j'}$, where the pair $(j,\, j')$ exists by the pigeonhole principle, and outputs $(p_j, p_{j'})$. Now, since $b_j = b_{j'}$ and $f(p_j), f(p_{j'}) \in \{u_0, u_1\}$, we know that $f(p_j) = f(p_{j'})$ if either ($f(p_j) = u_{b_j}$ and $f(p_{j'}) = u_{b_j}$) or ($f(p_j) \neq u_{b_j}$ and $f(p_{j'}) \neq u_{b_j}$). Hence:

$$\begin{aligned}
\mathbf{Pr}(f(p_j) = f(p_{j'})) &= \mathbf{Pr}(f(p_j) = u_{b_j}) \cdot \mathbf{Pr}(f(p_{j'}) = u_{b_j}) \\
&\quad + \mathbf{Pr}(f(p_j) \neq u_{b_j}) \cdot \mathbf{Pr}(f(p_{j'}) \neq u_{b_j}) \\
&= (1/2 + \delta(k))^2 + (1/2 - \delta(k))^2 \\
&= 1/2 + 2\delta(k)^2 \\
&> 1/2 + 2\mu(k)^2 \text{ (for all sufficiently large } k) \\
&= 1/2 + \epsilon(k)
\end{aligned}$$

where ϵ was assumed to be negligible. Thus $\mathcal{A}^{\mathtt{u}}$ breaks unlinkability, contradicting our assumption, and the result follows. □

9.2.7 Anonymity of users

Consider a sound pseudonym system that offers pseudonym unlinkability. The owner $u \in (U - \hat{U})$ of pseudonym p ($u = f(p)$) is hidden in the anonymity set $U - \hat{U}$ because, from $\mathcal{A}$'s point of view, any user in that set could potentially be the owner of p. The effective size of the anonymity set, however, depends on the probability distribution Δ according to which users were selected during the establishment of p. Using the information-theoretic anonymity metric and notation of [68, 184], this is given by

$$-\sum_{u \in U - \hat{U}} P_\Delta(u) \log_2 P_\Delta(u)$$

where $P_\Delta(u)$ denotes the probability that u is selected under distribution Δ. As shown in section 3.1 of [68], the value of the above expression is maximised if Δ is the uniform distribution, in which case the effective size of the anonymity set in which pseudonym p is hidden is $\log_2 |U - \hat{U}|$. It is worth observing that it makes sense to consider the anonymity of a user *while acting using a particular pseudonym*. In other words, two pseudonyms belonging to the same user may (and, in practice, almost certainly will) offer different degrees of anonymity for that user.

The above measure of anonymity only applies to a naive adversary; it only takes into account its *a priori* knowledge (i.e. the distributions Δ that it selected during the issuing of `runpeprot` queries). After observing the system for some time, in the sense of Game 4, $\mathcal{A}$ may decrease the unlinkability between pseudonyms. This decrease in unlinkability yields, for each pseudonym p, an *a posteriori* probability distribution Δ' over the set of users. This distribution effectively tells that $\mathcal{A}$ how likely it is for each user to be the owner of p. $\mathcal{A}$ is able to calculate Δ' using deductions that he can make from the scheme's soundness. While the distribution Δ' defines the (effective) size of the anonymity set in which a pseudonym is hidden, this does not necessarily mean that a reduction in unlinkability implies a reduction in anonymity in the theoretical definition of the term. Of course, in practice, any linking of pseudonyms is likely to lead to an increased risk of loss of anonymity because of 'out of scope' attacks. As a result, unlinkability is a property of great importance in its own right.

9.3 Summary

In this chapter we have introduced a complexity theoretic model for anonymous credential systems. We have formally defined the notions of pseudonym owner protection, credential unforgeability, credential non-transferability, and pseudonym unlinkability. A key challenge is thus to construct scheme(s) that meet the definitions in this model, and/or to prove, under appropriate assumptions, the security of existing ones. There is, however, room to refine and extend the model itself; determining the probability $\bar{P}$ by which colluding organisations should be bound when trying to correlate pseudonyms, given a specific history of events in the system, is clearly of importance. Naive strategies for computing $\bar{P}$ appear to be of exponential complexity. Hence, deriving efficient strategies for computing, approximating or bounding $\bar{P}$ is desirable.

It is hoped that further study of the model described above will enable better techniques to be devised for estimating the achievable degrees of unlinkability and anonymity, e.g. using the information-theoretic metrics that were recently proposed in [68, 184, 191]. This should provide further insight into the inherent limits of unlinkability and anonymity in credential systems. We believe that this will also provide insight into what such systems have to achieve in order not to be the weakest link in the context within which they will operate. An extended version of the model could capture additional properties of pseudonym systems, for example credentials that can be shown only a limited number of times, or a capability for anonymity revocation.

Another direction for future research is the analysis of real-world user behaviour in the context of pseudonym systems. This might lead to the description of strategies that users might follow, in a realistic setting, in order to maximise the unlinkability of their pseudonyms. Given the statistical properties of the context, this could also lead to descriptions of how long any given pseudonym can be kept before it should be renewed (if the context allows for this). Unfortunately, to the best of our knowledge and at the time of writing, there are no public deployments of anonymous credential systems and, therefore, this research has to wait.

Chapter 10

A peer-to-peer system that improves privacy of electronic transactions

Contents

This chapter describes a peer-to-peer system that helps protect against the attacks described in chapter 8. Moreover, the system provides real-time feedback to users of an anonymous credential system about the level of anonymity that they enjoy.

10.1 Introduction

In chapters 8 and 9 it was pointed out that colluding organisations can always link transactions that occur in a (sound) anonymous credential system by using the type of the credential involved. Chapter 8 elaborated on this attack in the case where credentials had to be checked for freshness. These attacks apply particularly to systems where credentials are relatively short-lived. It was therefore established that the degree of unlinkability that anonymous credential systems can provide is inherently limited, and depends on factors that may differ significantly between individual transactions. As users typically neither have any control

over, nor receive any feedback on, these factors, the real degree of unlinkability (and therefore privacy) they enjoy remains uncertain. In fact, in certain scenarios this uncertainty may lead to a false sense of privacy, since it is possible that the user's transactions are not unlinkable at all.

In this chapter we address these problems. The next section provides a motivating example and points out the problems in detail. Section 10.3 presents a peer-to-peer scheme that addresses the problems. Section 10.4 discusses security and privacy issues and section 10.5 describes a prototype implementation and its performance. Finally, section 10.6 concludes with directions for further research and development.

10.2 Motivation

Consider the following scenario. Alice is one of Igor's customers. As part of a loyalty reward scheme, she is entitled to a free cinema visit, and she obtains the relevant credential from Igor's website. Two hours later, Alice shows the credential to Victor who runs a cinema. In this scenario, **I**gor is the **i**ssuer and **V**ictor is the **v**erifier. Using Victor's website, she selects the movie she would like to see, i.e. "Fahrenheit 9/11", and prints her ticket[10.1]. Since the credential system used is anonymous, Alice expects that her privacy will be protected, i.e. that the system will prevent Igor from finding out which movie she went to see, and Victor from learning which of Igor's customers came to see "Fahrenheit 9/11".

Unfortunately, and as pointed out in chapters 8 and 9, even if the underlying cryptosystem flawlessly offers unlinkability, it may still be possible for Victor and Igor to link Alice's acts of credential issuing and showing, using timing information. The attacks we consider are made possible if one of the following conditions hold.

- It is known that most users show the 'cinema ticket' credential after some constant 'waiting' time (e.g. two hours). In this case Igor and Victor may conclude that the above issuing and showing acts correspond, with high probability, to the same user.
- Alice is the first and only person who obtains this ticket credential from Igor up to the time it is shown to Victor. In this case Igor and Victor reach the above conclusion

[10.1] The ticket contains a verification code that will allow the cinema staff to authenticate it as genuine.

with certainty.

Alice has no way of knowing how many others, if any, have obtained a similar credential from Igor. Without such feedback it remains unclear to Alice how unlinkable this particular pair of her transactions (issuing and showing) really is. The next section proposes a scheme that attempts to provide this type of feedback while offering resistance to the above timing attacks.

10.3 The scheme

This section presents the proposed scheme. A high level description is given first, in order to provide intuition for the protocol descriptions that follow.

10.3.1 High level description

The idea underlying our approach is based on the observation that users have to rely on each other in order for their interactions to remain unlinkable and, as a result, for their anonymity to be protected. In order to decrease the exposure to timing attacks we describe a scheme for the electronic world that attempts to model the following behaviour.

Users who intend to obtain a credential of some type first form a group; they 'meet' each other and wait. Only when there are enough users present, do they go ahead and obtain the credential. As long as this is done in a sufficiently synchronised manner, exposure to the timing attacks mentioned in the previous section is significantly mitigated. This is because:

- any given user knows that he is not the only one who wishes to obtain a credential of the specified type; moreover he has an estimate for the number of other users that wish to do so, and
- in scenarios with a constant waiting time between issuing and showing, the fact that all group members obtain the credential at approximately the same time means that they will also show it at approximately the same time (at least for the first time), thereby maintaining unlinkability.

It would be undesirable for users to rely on a third party to help them form groups, because this contradicts the threat model of anonymous credential systems (where it is assumed that any external organisations may collude against the user). Instead, users need to be able to form groups in an ad hoc manner. This service appears to be a natural peer-to-peer application, and our scheme follows this paradigm.

10.3.2 Roles

Three different roles must be supported in order for the scheme to work. The first role, namely that of a discovery server (DS), is not specific to a particular credential type; the DS role-holder provides a general supporting function for users of the scheme. The second role, that of group manager (GM), is specific to a credential type, and supports the formation of a group of users all wishing to gain a credential of this type. The third role is that of a group member (PM), wishing to obtain a credential in a way that provides additional anonymity protection.

10.3.2.1 Discovery Servers

As in any peer-to-peer scheme, users need a way of finding each other. For this reason, those who are willing to donate some of their spare bandwidth and computational power may run a DS, in a role similar to that present in many existing peer-to-peer systems. How the list of currently active DSs is communicated to users is outside the scope of the scheme, but this could be achieved using websites, dynamic updates, peer-to-peer messaging, or by some other means.

A user who wants to obtain a credential as part of a group, first needs to join such a group. If he does not know the address of any suitable GM (see below), he first has to ask a DS. To achieve this, he issues a 'discovery request' to the DS that contains descriptions of the credential type he wishes to obtain, the minimum and maximum acceptable size of the group he wishes to be part of, and applicable policy statements (see section 10.3.3). The DS searches its list of registered GMs and responds with the addresses of those that satisfy the given requirements. If no suitable GM is found, the user may either try a different DS, or establish his own GM service for the required credential type.

10.3.2.2 Group Managers

The GM service is also provided by a user willing to donate some of his spare resources. The lifetime of a GM, however, is expected to be much less than that of a DS because it typically ends when the group members obtain the credential of the required type. A GM is responsible for managing the assembly of a group of users all wishing to obtain a credential of a specified type. In particular, the GM's duties include (1) registering at one or more DSs, (2) admitting new users to the group, (3) providing feedback to existing members about users joining and leaving, (4) responding to explicit update requests from members, (5) informing members when the group is full and ready to obtain the credential in a maximally synchronised manner, and (6) deregistering from DSs (which should take place as soon as the function of the GM is complete).

The GM of our scheme additionally acts as an anonymising proxy through which all group members obtain their credentials (i.e. the scheme provides an anonymous user-to-issuer channel). Issuers only see the GM's network address. As a result, even if verifiers see a PM's real network address, unlinkability is not compromised through communications with the GM.

10.3.2.3 Group members

Any user who wishes to obtain a credential using this scheme, and who does not wish to act as a GM for this credential type, must become a PM by joining an appropriate group. The PM joins a group by sending a 'join request' to a suitable GM. The GM then informs the PM how many other PMs are currently in the group and, during the subsequent lifetime of the group, notifies the PM when other PMs join or leave the group. Finally the GM notifies the PM when it is time to obtain the credential. Once this latter stage is reached, the PM should then obtain the credential from the issuer, using the GM as a proxy. PMs do not have to blindly trust the GM. They may, for example, ping each other periodically in order to establish liveness and trust in the GM.

10.3.3 Policies

As part of the above scheme, GMs and PMs follow and enforce policies. GMs report their policies to DSs during registration and include them in their responses to join requests. A GM's policy not only includes the size of the group it serves, but also may include restrictions that are required to hold for the group as a whole. For example, a GM that wishes to establish a group of 15 PMs before helping them to obtain credentials may require that the PMs in the group belong to at least five different administrative domains and three different jurisdictions. In general, a GM will refuse to admit new PMs if their presence in the group may prevent the GM's policy being satisfied.

PMs may similarly place restrictions on the groups they join. These PM policy statements are sent to the DS as part of a discovery request, and the DS may choose to only report back to the PM those GMs that are consistent with the expressed policy. The PM will also check the policy reported by the GM during the joining process against its own policy. The PM can also check the group composition against its policy statements on an ongoing basis, as the group forms. If a PM realises that its policy, at any point, is no longer satisfiable, then it may withdraw from the group.

Fur the purposes of the scheme, the policy for each of the communicating entities is assumed to be a collection of statements expressed using some generally agreed syntax. A policy field is included in certain of the messages of the scheme. However, it is neither necessary nor desirable to send the whole collection of policy statements belonging to an entity in every protocol; instead, a subset (or 'view') of the set of policy statements is selected according to the circumstances and purpose of the protocol.

10.3.4 The protocols

There are nine protocols in our scheme: `Echo`, `Register`, `Discover`, `Join`, `Leave`, `SingleUpdate`, `Update`, `Deregister`, and `Run`. These are now described.

10.3.4.1 `Echo`

This protocol is executed between a pair of peers. Its purpose is to allow the initiator to establish the liveness of the responder and to estimate the latency between them. It operates as follows.

`Initiator⟶Responder: Echo Request`

`Responder⟶Initiator: Echo Reply`

The `echo request` message contains the initiator's timestamp, indicating the time at which the message was sent. The responder echoes the timestamp back to the initiator in the `echo reply` message. The initiator then estimates the latency based on the difference in time between receipt of the message and the timestamp. The protocol is repeatedly executed at regular time intervals, defined by the initiator's policy and preferences. The allowed (initiator, responder) role pairs for this protocol are as follows.

- (PM, PM): A PM may run the protocol with another PM in order to establish that the latter exists. Successful completion of the protocol may serve as evidence that the GM is not misbehaving (by reporting non-existent PMs).
- (GM, PM): A GM should run the protocol with all PMs in its group in order to detect disconnected or failed network nodes. If no response to an `Echo Request` is received within a specific timeout period, the PM is removed from the group, and the remaining PMs are notified using the `SingleUpdate` protocol (see below). The GM also records and keeps a running average of the latency for each PM.
- (PM, GM): As GMs may also unexpectedly disconnect from the network, PMs are required to run this protocol with the GM. If no response is received within a specified timeout period, the PM assumes that the group no longer exists and may try to join another such group.
- (DS, GM): A GM registers at one or more DSs at the beginning of its lifetime, but may fail to deregister once it has completed its task. A DS executes this protocol in order to detect and remove 'dead' GMs from its lists.
- (GM, DS): A GM may execute this protocol in order to detect failed DSs.

10.3.4.2 `Register`

This protocol is executed between a GM and a DS. Its purpose is to enable the GM to register with the DS (so that it can 'be discovered' subsequently) and operates as follows.

`GM`⟶`DS: RPolicy`

`DS`⟶`GM: Success` or `Failure`

The GM sends (an appropriate view of) its policy to the DS. Under normal circumstances, the DS adds the GM to its list of active GMs and responds with a `success` message. However, if, for whatever reason, the DS is unwilling to do so (e.g. the GM is known to be a misbehaving node), then it responds with a `failure` message. In this case the GM may wish to register with another DS.

10.3.4.3 `Discover`

This protocol is executed between a PM and a DS. Its purpose is to enable the PM to obtain the network addresses of GMs that have been set up to establish credentials of the desired type, and that satisfy the PM's policy. It operates as follows.

`PM`⟶`DS: DPolicy`

`DS`⟶`PM: GMList`

The PM sends a discovery request message to the DS, specifying (an appropriate view of) its policy and the credential type it wishes to obtain. The DS responds with a list of GMs that satisfy the PM's requirements. If the DS cannot find a suitable GM the list is empty.

10.3.4.4 `Join`

This protocol is executed between a PM and a GM. Its purpose to enable for the PM to join the group established by the GM. It operates as follows.

`PM⟶DS: JPolicy`

`DS⟶PM: (GPolicy || GList)` or `Failure`

First, the PM sends (an appropriate view of) its policy to the GM. If the group is not full, and if the policies match, the GM admits the PM to the group by responding with the (appropriate view) of its own policy and the list of PMs that are already part of the group. If, for any reason, the GM decides not to admit the PM, it responds with a suitable error message. In this case the PM may wish to join another group.

After a sufficient number of PMs has joined the group, the GM runs the `Deregister` protocol with all DSs it was registered with, and the `Run` protocol with all the PMs in its group (see below).

10.3.4.5 `Leave`

This protocol is executed between a PM and a GM. It allows the PM to announce that it wishes to leave the group. It operates as follows.

`PM⟶GM: Leave`

After receiving a `leave` message from a PM, the GM removes it from the group membership list and notifies the remaining PMs of the change using the `SingleUpdate` protocol (see below).

10.3.4.6 `SingleUpdate`

This protocol is run between a GM and all the PMs currently in the group. The purpose of the protocol is to let the PMs know when a PM joins or leaves the group.

`GM⟶PM: SingleUpdate`

The message contains the PM address and the action (joining or leaving) taken by the PM in question.

10.3.4.7 `Update`

This protocol is periodically executed between a PM and its GM. It allows the PM to explicitly request the current member list of the group, and operates as follows.

`PM⟶GM: Update Request`

`GM⟶PM: Update Reply`

The `update reply` message contains the full list of all PMs currently in the group. The protocol is necessary to re-synchronise PMs that have missed `SingleUpdate` messages. The protocol also allows each PM to check that it has not been evicted from the group (for example, because of a lost `echo reply` message).

10.3.4.8 `Deregister`

This protocol is executed between a GM and a DS. Its purpose is to allow the GM to announce to the DS that it is no longer acting as a group manager and operates as follows.

`GM⟶DS: Degerister`

The `deregister` message contains a description of the reasons for deregistration, typically that the GM has successfully completed the credential establishment process for its group members. The DS removes the GM from its list.

10.3.4.9 `Run`

This protocol is run between a GM and a PM. Its purpose is to allow the GM to announce that the group is now full and that the group members should now each obtain a credential. It operates as follows.

`GM⟶PM: Run`

`PM⟶GM: Acknowledgement`

The GM must send the `run` message to all the PMs in the group in a maximally synchronised manner. The `acknowledgement` message serves as a final confirmation that the PM is willing to obtain the credential. After the receipt of this acknowledgement, the GM connects to the issuer and acts as a proxy for the PM for the remainder of the communication.

Comment: Ideally, the `SingleUpdate` and `Run` protocols should be implemented via a broadcast technique that allows the GM to send the message to all its PMs simultaneously. In a routed network (such as the Internet), however, this it not possible and could be replaced by multicast or dedicated unicast messages for each PM. In this case, the GM should take into account the individual network latencies that were recorded by the `Echo` protocol, in order to maximise the synchrony with which the PMs act.

10.4 Security and privacy

Given that this is a service that is offered at the network level, the scheme does not depend on the underlying anonymous credential system. Therefore the above scheme does not affect the security of the underlying credential system (in terms of credential unforgeability and similar security notions — see Chapter 9). In the ideal case, where all users interested in a credential of a given type form a single group, and then obtain the credential together in a perfectly synchronised manner, the degree of unlinkability [191] of the transactions of a single user is maximised in the face of the timing attacks mentioned in section 10.2. Further, users learn the size of the group, and can therefore estimate this degree of unlinkability.

Unfortunately, unlinkability may be compromised in the presence of an adversary that colludes with organisations. An omnipresent adversary, for example, that controls all network communications, can impersonate DSs, GMs and PMs; it can therefore easily mislead a user into believing that he is part of a group of any particular size while, in fact, he may be alone. While this does not necessarily mean complete loss of unlinkability (other legitimate users may still obtain a credential of the same type from the issuer), the scheme — like all anonymity schemes — collapses in the presence of such an extremely powerful adversary. Of course, such a powerful adversary may be very difficult to implement in practice.

A possibly more realistic adversary could compromise or introduce its own DSs, GMs and PMs. This attack is equivalent to the 'Sybil' attack that is always possible in de-

centralised peer-to-peer networks where there is no reliable method of establishing node identities [71]. Indeed, it is arguable that protecting against Sybil attacks requires a level of linkability for network nodes, so that, where necessary, two 'identities' of the same entity can be linked. An adversarial DS can gather the network addresses of users that express interest in obtaining a credential of a particular type. This information, however, would in any case be disclosed to issuers directly if the scheme was not in use. An adversarial GM is more serious as it learns the network addresses of all PMs in the group. Further, it can report the presence of nonexistent PMs or initiate the credential issuing procedure prematurely. Such behaviour, however, can be detected by PMs because they may ping each other directly and, if not satisfied, may leave the group. An adversarial PM also learns the network addresses of other PMs in the group. Additionally, it reduces the group's effective size by one and therefore gives the other PMs a false sense of privacy. The careful management and use of policies mitigate the exposure to this risk. A policy, for example, that requires group members to originate from at least x different network locations implies that the adversary must be present at $x-1$ locations in order to fully compromise the anonymity of victim PM, assuming that network locations cannot be spoofed.

There are several ways for a user to reduce its exposure to potentially adversarial DSs and GMs. Firstly, DS lists should be updated frequently, and each query should ideally be sent to a different subset of DSs. This will decrease the probability of repeatedly choosing adversarial DSs. Secondly, users can maintain a list of the GMs that were returned by different DSs. If a DS reports the same GM(s) in different time periods, then this may be an indication that both are adversarial and can thus be blocked. Thirdly, a reputation scheme could be used in order to block untrustworthy peers.

Finally, the adversary could perform traffic analysis in order to find out the network addresses of PMs. In this respect our scheme is similar to the MorphMix anonymous peer-to-peer communication scheme [178]; the dynamically changing topology and short lifetime of groups requires an almost omnipresent adversary in order to carry out this attack effectively over a long period of time.

In any case, if an adversary that colludes with organisations and learns the network addresses of PMs is a concern, it is advisable that users show their credentials to organisations using some other anonymous channel. In any event, use of such a channel during credential showing is highly recommended in the case of multiple-show credentials, in order to avoid

the possible linking of different showing events (rather than linking the issuing and showing acts).

10.5 Implementation

This section describes our prototype implementation of the scheme, which was written in Java 1.4.2. It has been successfully tested under Windows, Red Hat Linux and Knoppix. Each peer role is implemented as a service that runs over customisable TCP ports. All protocols were implemented over TCP (rather than UDP) in order to avoid the complexity of dealing with lost or out-of-order datagrams. A dummy credential issuer was implemented in order to test the proxy functionality and measure the arrival times of credential issuing requests. The source code of the implementation can be found in Appendix B. Due to the limitations of our test environment (both technical and geographical), the use of group restrictions in policies, such as specifying a minimum number of represented subnets or jurisdictions, were not implemented.

In order to give an indication of performance, an experiment was conducted. It involved forming a group of nine PMs, all of which then obtained a credential from an issuer. The PMs were managed by one GM which was registered at a single DS. Four machines were used in the experiment. Their respective configurations and connection types were as follows.

1. Pentium III 500 Mhz, Windows 2000, DS, Dummy Issuer, 3 PMs, 10Mbp/s (ethernet).
2. Pentium III, 1.2 GHz, Windows 2000, GM, 2 PMs, 52Kbps (modem).
3. Pentium IV, 2.4 GHz, Knoppix, 2 PMs, 10Mbps (ethernet).
4. Pentium IV, 2.4 GHz, Knoppix, 2 PMs, 10Mbps (ethernet).

Note that, in order to artificially cause non-trivial network delays, the GM was running on machine 2, i.e. the machine with the slow modem connection. Also, machines (3) and (4) were located on a different subnet to machine (1). Figure 10.1 shows the time it took for the views of PMs to converge as new PMs joined the group. In particular, the time represents the number of milliseconds between the instant the GM received the join request of a new PM until the last existing PM received notification of this event. Observe that this

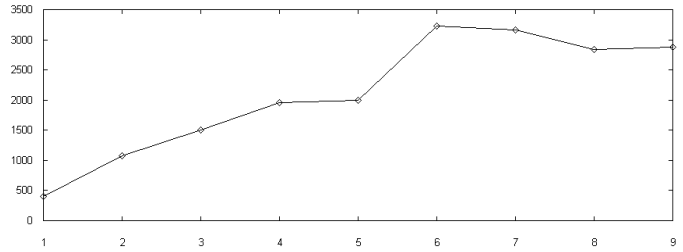

Figure 10.1: Convergence delay for a group of nine peers

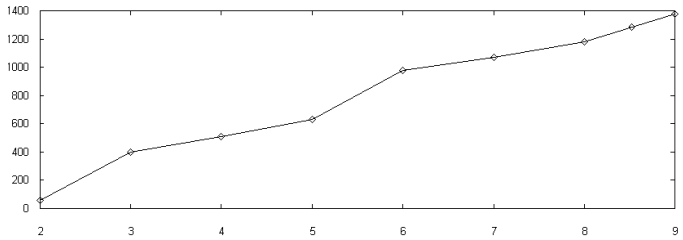

Figure 10.2: Arrival time of issuing requests for a group of nine peers

delay is determined by the PM which has the slowest connection to the GM; most PMs will have updated their views faster than that.

Figure 10.2 depicts the degree of synchronisation achieved. In particular, the vertical axis shows the number of milliseconds between the arrival of credential issuing requests at the issuer; all nine issuing requests arrived within less than 1.4 seconds. Given the amount of time it currently takes to actually issue an anonymous credential (which is typically in the order of several seconds [38]), this level of synchronisation is likely to be adequate.

Finally, Figure 10.3 depicts the user interface of a GM. It consists of a simple, user friendly progress bar which shows dynamically the number of users in the group. Below that we see the list of (the network addresses of) the DSs the GM is currently registered with, along with their respective latencies as recorded by the `echo` protocol. The first number counts the number of pings exchanged, the second is the latest latency, while the third represents a moving average. Further below we see the list of (the network addresses of) the current PMs, again with their respective latencies. The interface for a PM consists of a similar progress bar and, optionally, the list of current PMs in the group.

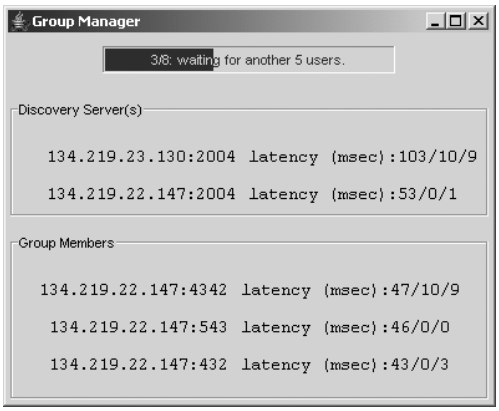

Figure 10.3: Screenshot of Group Manager

10.6 Summary

This chapter introduced a new peer-to-peer application that improves user privacy properties of anonymous credential systems. Peer-to-peer based privacy enhancing technologies that have been proposed and developed in the past, such as that described in [178], focus on hiding the user's real network address from servers; the main focus of our scheme, by contrast, is the synchronisation of events in an anonymous credential system in order to protect against timing attacks that would compromise the unlinkability of those events. Moreover, the scheme provides feedback to users that enables them to quantify the obtained degree of unlinkability. We also presented a prototype implementation of the scheme and described its efficiency.

Several research and implementation issues remain open. It would be interesting to investigate the feasibility of integrating our scheme with an implementation of a 'real' anonymous credential system, for example the idemix anonymous credential system [40]. Such an integration would unify user interfaces and enable testing in a more realistic environment. Integrating our scheme with a general peer-to-peer anonymous communication system, such as [178], is another possibility. Such an integration, however, only makes sense if a real anonymous credential system is already running underneath.

Refining the protocol implementation could improve the efficiency considerably. In particular, using UDP rather than TCP for certain messages would avoid the TCP connection setup/teardown overheads that the current implementation has to bear. However, this will introduce more complexity, as the application will then need to perform end-to-end transport

management. It may also be worth investigating the possibility of coupling our scheme with a suitable reputation scheme, in order to detect and block malicious peers more efficiently. Finally, it remains to conduct a rigorous analysis of the unlinkability offered by anonymous credential systems in general, and in particular when used in conjunction with our scheme.

Chapter 11

Introduction to privacy-aware single sign-on

Contents

This chapter introduces the notion of a privacy-aware SSO system. It also examines different possible architectures for such schemes, and considers both their feasibility and the extent to which 'privacy-awareness' can be achieved.

11.1 Introduction

In the context of SSO it is sometimes desirable, from a privacy perspective, to prevent different SPs from linking different identifiers that belong to the same user. This requirement should be met even if SPs cooperate and actively try to correlate the identifiers. An SSO scheme that meets this requirement, i.e. that prevents such correlations taking place, is said to be 'privacy-aware'. It is worth noting that, if an SSO scheme is proxy-based, we call it privacy-aware only if it prevents this identifier linking even if SPs cooperate with the operator of the proxy. In other words, SPs are not only allowed to cooperate with each other, but also with the proxy, if one is present.

Of course, there may be ways in which cooperating SPs/proxy operators may link user identifiers, even if the SSO scheme in use is privacy-aware. Traffic analysis, linking of user attributes, and the analysis of his behaviour or location are examples of strategies that may be employed by cooperating SPs and proxy operators when attempting to correlate identifiers. These types of attack, however, occur at a level of abstraction that is different from that at which the SSO scheme operates. They therefore lie outside the scope of the system. If one wishes to protect against them, other mechanisms should be used in parallel with the SSO scheme. If, for example, traffic analysis would enable SPs to link different identifiers, a mechanism that provides an anonymous user-to-SP channel should be used in conjunction with the scheme. A privacy-aware SSO scheme could be augmented with functionality that protects against these types of attack. As mentioned in section 2.1.4, such an augmented system could then be considered to be an Identity Management system.

In this chapter, we examine each of the four types of SSO system architecture that were introduced in chapter 3 with respect to the feasibility and extent to which privacy-awareness can be achieved. We also revisit two concrete and well-known SSO schemes with respect to privacy-awareness.

11.2 Privacy-awareness in different architectures

This section examines the four types of SSO scheme, namely local pseudo-SSO, local true SSO, proxy-based pseudo-SSO and proxy-based true SSO, with respect to the feasibility and extent to which privacy-awareness can be achieved.

11.2.1 Local pseudo-SSO

Recall that in a local pseudo-SSO scheme the AS is running on the user machine. After authenticating the user locally, it automatically executes SP-specific user authentication mechanisms on the user's behalf. As mentioned in section 3.4.1, under pseudo-SSO it is the SPs that determine the format and sometimes the content of the user identifier. Co-operating SPs may thus force the user to use an identifier that is unambiguously linked to his real identity (e.g. his national insurance or his credit card number) and can, in this way, trivially link his transactions.

Many SPs specify the format but not the content of the user identifiers. They might, for example, expect users to choose their own usernames or require them to log in using an e-mail address. In these cases, a user who is interested in keeping his identifiers unlinkable should choose them in a way that obscures any conceivable correspondence. Unfortunately, in practice, users cannot be relied upon to generate such identifiers. A local pseudo-SSO scheme could be augmented to choose such identifiers on behalf of the user at registration time, and implement rules ensuring that no obvious correspondence exists between any two identifiers. The effectiveness, however, of this measure depends on the format that different SPs impose on user identifiers. Thus the extent to which a pseudo-SSO scheme can be privacy-aware depends on issues outside the scope of the scheme itself.

11.2.2 Local true SSO

In a local true SSO scheme, the AS again runs on the user machine, but here all SPs follow the same protocols for the purposes of user identification. This means that the identifiers have a uniform format throughout the system. A local true SSO scheme can therefore be privacy-aware, if properly constructed. The scheme described in section 6.3.2 is an example of a privacy-aware local true SSO scheme. (The privacy awareness of that scheme was analysed in section 6.4.2.)

Unfortunately, if a local true SSO scheme is not privacy-aware by design, it is not straightforward to modify it to ensure that it does possess this property. In order to see this, consider the following example of a PKI-based local true SSO scheme. Each user has a key pair for a signature scheme, the public part of which is certified by a system-wide CA and the private part of which is kept in a smartcard. In order to log into an SP, a challenge message sent by the SP to the user, e.g. a random number, must be signed using the user's private key. The signature is sent, along with the user's public key certificate, to the SP who checks them for correctness. For the signature to be generated, the user's smartcard has to be present. Thus, this simple scheme provides user authentication by means of proof of possession of the smartcard. The scheme described in chapter 5 is based on this idea. It is not privacy-aware since the same public key is sent to all SPs. (The user identifier, another field in the public key certificate, is also unique throughout the system.) Moreover, the CA knows the identifiers of all users in the system. Making this scheme privacy-aware would involve the generation of unlinkable user identifiers, preventing the CA from learning these

identifiers while, at the same time, providing assurance to SPs that users are in possession of genuine cards.

It seems that one has to resort to fundamentally different cryptographic primitives, such as those on which anonymous credential systems are based, in order to achieve privacy-awareness. Of course, this remark applies to all (local and proxy-based) true SSO schemes that are not themselves based on such primitives.

11.2.3 Proxy-based pseudo-SSO

In a proxy-based pseudo-SSO scheme, the proxy executes SP-specific authentication methods on the user's behalf. This means that, by necessity, the proxy has access to all the user's identifiers. Therefore, privacy-awareness cannot be achieved. Typically, however, it is not an issue if the proxy knows the user's identifiers, as it is trusted to perform authentication on his behalf, a task that is usually perceived as more sensitive. In this scenario it is still desirable, from a privacy perspective, to prevent SPs from linking different identifiers, albeit without being allowed to cooperate with the proxy. A notable advantage of the proxy-based pseudo-SSO architecture in this respect is that the proxy can provide an anonymous user-to-SP channel as an additional service, thereby protecting against traffic analysis and similar attacks that would enable the SPs to link identifiers whose correspondence would be otherwise obscured. The scheme described in chapter 7 implements precisely these services. Had the scheme been a local one, such a service would have to be provided by an additional entity.

11.2.4 Proxy-based true SSO

A proxy-based true SSO scheme can be designed to be privacy-aware, since all SPs are required to follow the same protocols. In such a scheme, a user would use an identifier (i.e. a pseudonym) p_0 with the AS (which runs on the proxy), and separate identifiers $p_1, p_2, \ldots, p_n$ with the n SPs in the system that they have a relationship with. All $n+1$ identifiers would need to be unlinkable to one another as well as to the identity of the user. The user would authenticate himself to the proxy under p_0 and, if successful, obtain an 'authentication assertion' from the AS. He would use this assertion in order to log into one of the $n-1$ SPs using the appropriate identifier.

Intuitively, the authentication assertion should not contain the user's identifier p_0, but the user should, at the same time, only be able to use it with one of his own identifiers, i.e. $p_1, p_2, \ldots, p_n$. More precisely, the assertion may 'contain' p_0 as long as it is not revealed to SPs during the login process. It is also clear that new identifiers have to be established in the system in a uniform way that ensures that (a) they are unlinkable even if SPs and the proxy cooperate, and (b) no user can log into SPs under pseudonyms that were not established by himself. As shown in the previous chapters, anonymous credential systems provide such services. In the following chapter we examine ways to construct proxy-based true SSO schemes using such systems.

It is worth noting here that, unfortunately, the proxy of a privacy-aware SSO scheme cannot be used to provide an anonymous user-to-SP channel. This is because by doing so (and by cooperating with the SPs) it could trivially correlate the user's identifiers using traffic analysis, thereby defeating the very goal of privacy-awareness.

11.3 Privacy-awareness in existing schemes

In this section we revisit two well-known proxy-based true SSO schemes, namely Microsoft Passport and the Liberty Alliance, and briefly examine them with respect to privacy-awareness.

Microsoft Passport: As mentioned in section 3.5.3, Microsoft Passport users are identified throughout the system using a unique 64-bit number, chosen by the proxy (i.e. Microsoft). Therefore, any subset of SPs can cooperate in order to combine their respective information about any given user, even without help from Microsoft.

Liberty Alliance: As mentioned in section 3.5.2, the Liberty Alliance specification requires the proxy to choose a different identifier for each user/SP pair. This means that different SPs know a given user under different identifiers. However, when the user obtains an authentication assertion from the proxy, he has to state SPs he wishes to use it with, so that the correct identifier is encoded into the assertion. As a result, if the proxy collaborates with the SPs, user pseudonyms can be linked. Hence, an SSO scheme that conforms to the specifications is not privacy aware. Under the assumption, however, that the proxy is honest and does not cooperate with SPs, the user's identifiers appear unlinkable to any subset of

cooperating SPs. If, of course, the proxy is malicious, it can easily encode information into a user's identifiers that would enable anyone to link them.

In order for the Microsoft Passport or the Liberty Alliance schemes to become privacy-aware, a major redesign would be needed. Their operation would then require the installation of special software on the user's machine to support the required protocols, data structures and privacy-preserving cryptographic primitives. However, one of the design goals for these schemes was that they should work with already deployed web browsers while explicitly avoiding the need for a user to install additional software [167]. It is therefore perhaps fair to note at this point that both schemes essentially achieved this goal, something that would have been impossible if they were designed to be privacy-aware in the first place.

11.4 Summary

This chapter introduced the notion of privacy-awareness for an SSO scheme. A scheme that is privacy-aware prevents SPs in the system from linking the identifiers of a given user. A proxy-based scheme is also required to protect against collaboration between SPs and the proxy. To the best of the author's knowledge, no existing real-world SSO scheme is privacy-aware in this sense.

It was shown that local pseudo-SSO schemes can only hide the correspondence between a user's identifiers to the degree that the SP-imposed format of these identifiers allows. While proxy-based pseudo-SSO schemes cannot, by definition, achieve privacy-awareness, true SSO schemes, both local and proxy-based, can, as long as they are properly designed and constructed. Finally, it was shown why Microsoft Passport and the Liberty Alliance, two well-known proxy-based true SSO schemes, fail to possess the privacy-awareness property.

Chapter 12

Constructing privacy-aware single sign-on systems

Contents

This chapter shows how to combine proxy-based true SSO with anonymous credentials in order to achieve privacy-aware user authentication in an open environment. Issues that arise are also discussed.

12.1 Introduction

To the best of the author's knowledge, virtually all existing real-world true SSO schemes, including those described in part I of this thesis (except the scheme described in section 6.3.2), do not prevent the operator of the proxy from working out the user's identifiers at the SPs of the system. In these systems, the user either has a single system-wide identifier, as is the case with a traditional PKI (see section 1.1.5), Kerberos (see section 3.5.1), and Microsoft Passport [144] (see section 3.5.3), or, if different SPs know the user under different identifiers, the proxy knows these identifiers. (For example, the Liberty Alliance specifications require the

proxy to generate SP-specific identifiers for the user — see section 3.5.2). From a privacy perspective, this situation is undesirable. Further, in existing proxy-based SSO schemes, the proxy can trivially impersonate users to all the SPs in the system. This situation is undesirable from a trust management perspective.

In this chapter, we describe the operation of a proxy-based true SSO scheme without these deficiencies, along with issues that arise in this particular setting. By contrast with the scheme described in section 6.3.2, the scheme described here does not require tamper-resistant hardware. In line with the definition given in Chapter 11, we call this SSO scheme 'privacy-aware', meaning that it hides the identifer-to-user mapping from the SPs in the system. In other words, the mapping between users and identifiers is not revealed to the SPs, even if they collaborate (and, in the case of proxy-base schemes, even if the SPs collaborate with the operator of the proxy).

12.2 Motivation

The previous chapter discussed the feasibility and extent to which privacy-awareness can be achieved in any of the four types of SSO scheme that were identified in chapter 3. It was shown that, while proxy-based pseudo-SSO schemes cannot, by definition, be privacy aware, local pseudo-SSO schemes can, if either the user or local software generates unlinkable pseudonyms. However, restrictions on the format and content of pseudonyms, imposed by SPs, may limit the ability to generate unlinkable pseudonyms. Moreover, users are unlikely to be able, or willing, to generate such pseudonyms.

True SSO schemes, on the other hand, have the potential to be privacy-aware in the above sense. In order to motivate the construction of privacy-aware true SSO schemes, we briefly revisit some of the advantages of true SSO over pseudo-SSO schemes.

- A global policy can be formulated, published and enforced by the AS operator. This cannot be achieved in a pseudo-SSO scheme, since in such a scheme each SP uses its own user authentication mechanism; hence, the authentication mechanisms used by the SPs are essentially independent of each other. Thus, no precisely defined trust relationships exist with the AS operator, and therefore no global policy can be formulated.

- Several user authentication mechanisms can be supported in parallel. The AS can offer these as a service to both users and SPs. Under pseudo-SSO this possibility does not exist for the same reasons as above.
- An SP may change its preferred user authentication mechanism, without having to change its infrastructure, protocols and interfaces. As long as the AS offers the desired mechanism, all the SP has to do is to publish its preference.
- User mobility is preserved. The user can authenticate himself from anywhere in the network to the AS and use the SSO scheme. However, the chosen user authentication mechanism may impose certain restrictions on this.

Note that the last point applies only to a proxy-based scheme. In the sequel we focus on an SSO system where a proxy authenticates users without learning their SP-specific identifiers. We call such a system a privacy-aware proxy-based true SSO (PAPTS) system. Of course, there may be more than one proxy in a given network; this would result in an architecture similar to that of the Liberty Alliance specifications. For simplicity of exposition, however, in the sequel we focus on the case where only one proxy exists.

12.3 Privacy-aware proxy-based true SSO systems

The construction of a PAPTS scheme using an anonymous credential system appears to be straightforward. The idea underlying the constructions given below is to have the AS issue an anonymous credential to the user that encodes his authentication status, and the user subsequently shows the credential to SPs in order to access a protected resource. We identify below two different approaches to the construction of a PAPTS scheme using this general approach. We also discuss a number of issues that arise in this special application of an anonymous credential system.

Before describing the constructions, is is necessary to introduce the notion of an *authentication context* [129]. An authentication context is a description of the means used to authenticate the user to the proxy. It includes all the information that an SP is likely to require in order to make a decision when the user attempts to access a protected resource using an SSO scheme. The amount and nature of this information depends on the environment and technical issues, but may also depend on legislation and culture. Each of the user

authentication mechanisms of the SSO schemes in part I of this thesis has its own context. This is implicitly defined by the scheme and the environment in which it is deployed. As in this chapter we use an arbitrary anonymous credential system along with an arbitrary user authentication mechanism, we need to explicitly define what makes up an authentication context; this is because the context determines the type of credential that will be issued in the system.

The following is a list of the types of information that together define an authentication context.

- a description of the authentication mechanism used (proof of knowledge of a secret, biometrics, or proof of possession of a hardware token),
- a description of the protocols and/or cryptographic primitives that are used for authentication (e.g. password hash, Diffie-Hellman key agreement, RSA signatures, etc.),
- a description of the initial user registration procedure at the AS (e.g. face to face, e-mail verification, shared secret),
- a description of the protection mechanisms in place at the AS (e.g. restricted physical access, third-party security accreditation),
- a description of the protection mechanisms possessed by the user (e.g. encrypted files, smartcard), and
- a description of service level or liability agreements that apply.

We now introduce notation that will be used for the description of our PAPTS constructions. In our scenario we suppose that a user U wishes to log into two SPs, denoted A and B. Before the user can use the system for SSO, he needs to establish a pseudonym with the AS (i.e. the proxy) and the two SPs. In both constructions we require that he does so using the pseudonym establishment protocol of the underlying anonymous credential system. We denote the established pseudonyms by p_{AS}, p_A and p_B respectively. Moreover, the user and the AS must establish at least one authentication context under which the user can authenticate himself to the AS. The identifier under which the user authenticates himself is denoted U_{ID}. A high-level description of the SSO protocol in our constructions for a PAPTS scheme follows.

1. U $\leftrightarrow$ AS: negotiation of authentication context c
2. U $\rightarrow$ AS: authentication of U_{ID} to AS under c
3. U $\leftarrow$ AS: issue credential of type $t = (c, \tau)$ under p_{AS}
4. U $\rightarrow$ A: show credential of type t under p_A
5. U $\rightarrow$ B: show credential of type t under p_B

The credential type t encodes a combination of the authentication context c and a timestamp τ that represents the instant in time when authentication (step 2) took place. Observe that a user authenticates himself to the AS under the identifier U_{ID}, obtains the credential under the pseudonym p_{AS}, and shows it to SPs A and B under the pseudonyms p_A and p_B respectively. We now examine two different approaches to combining an authentication context with an anonymous credential system.

12.3.1 A general privacy-aware SSO scheme

In the first scheme, the authentication context remains decoupled from the anonymous credential system. This means that any number of user authentication mechanisms may be employed by the AS. Before the user can authenticate himself he has to establish at least one such context with the AS. In general, several contexts may be established by a user; this is why the first step of the protocol (step 1) is the negotiation of the context c that is to be used subsequently. The format of the identifier U_{ID} under which the user authenticates himself (step 2) depends on the negotiated context c. Unlike a pseudonym used in an anonymous credential system, the identifier U_{ID} might or might not bear a connection to the user's real identity; it could be a random number, a nickname, an e-mail address, a social security number, a biometric, the user's real name, etc.

In this setting it is possible that the AS learns the user's real identity as a result of establishing an authentication context, although not all contexts may require the AS to know the user's real identity. As an example of a situation where the AS will have to know the user's real identity, a user may have established a 'strong' context for which he was obliged to appear in person at the AS with his passport. Authentication under this context may be required by high-risk scenarios, for example online banking or electronic government transactions. At the same time, the user may have established a 'normal' context, for example

by demonstrating the ability to receive e-mail at a particular address. This context may be sufficient for lower-risk operations, such as online purchases or logging into electronic forums.

A user may not only establish different authentication contexts with the AS, but, assuming that this is allowed, he may establish multiple identifiers U_{ID} for a given context. Each established indentifier U_{ID} (whether or not in the same context) has to be unambiguously linked to one of the pseudonyms that the user has established with the AS. This binding is maintained by the AS, as it has to issue credentials under the correct pseudonym p_{AS} (step 3). In order to keep identifiers separate, a user could establish a new pseudonym p_{AS} for each AS identifier U_{ID}. However, in certain situations this may be an avoidable overhead. For example, if two identifiers U_{ID} can be trivially linked to each other, or if the user does not mind the AS knowing that both correspond to the same user, the same pseudonym p_{AS} may be associated with both.

12.3.2 A simplified privacy-aware SSO scheme

The PAPTS scheme described in the previous section essentially required two systems to run in parallel, namely a series of user authentication mechanisms and an anonymous credential system. In order to use that scheme, the user has to establish at least two identifiers with the AS, namely one for authentication and one for credential issuing. In this section we describe a special case of the previous scheme that is less complicated.

A user of a pseudonym system typically possesses a secret associated with each of his pseudonyms. This enables him to prove, for the purposes of credential issuing and showing, that he is the legitimate owner of the pseudonym in question. Thus, in the context of PAPTS, the pseudonym system already provides a protocol for authenticating the holder of a pseudonym to the proxy and the SPs. Rather than adding complexity to the system, in certain situations it may be reasonable to use this protocol for user authentication at the proxy (step 2 in section 12.3). In such a setting, the user does not have to establish a separate U_{ID} or, more precisely, in this setting $U_{ID} = p_{AS}$. As p_{AS} is established using the pseudonym system protocols, the pseudonym does not have any connection to the user's real identity. Therefore, this approach supports situations where there is no need for the AS to know the user's real identity.

A question that arises naturally in this setting is the following. Since, in a given pseudonym system, users are required to authenticate their pseudonyms using the system-specific protocol, can the proxy then offer multiple authentication contexts c?

Indeed, the pseudonym authentication protocol of the pseudonym system may limit the types of authentication mechanism a PAPTS system can offer, as all such mechanisms must conform to the pseudonym system protocols. In other words, in this simplified version of PAPTS, the AS can no longer offer arbitrary user authentication mechanisms, but only those that can be implemented as part of the pseudonym system.

Even under these constraints, it is still possible to construct different user authentication contexts. If, for example, the protocols require the user to prove knowledge of a secret value, it is potentially important for the SP to be able to determine where that value is stored and how it was generated. A smartcard-based mechanism is different from a setting where the value is simply stored in a computer file. Similarly, a mechanism where the secret was generated using high-entropy randomness is different from a mechanism where the randomness was derived from a password. Although the authentication protocols may be identical, the associated user authentication contexts are different.

12.4 Issues

This section discusses issues that arise in the PAPTS systems described above. In particular, the next subsection identifies an important advantage of the PAPTS schemes constructed here over 'traditional' proxy-based SSO schemes, and sections 12.4.2, 12.4.3 and 12.4.4 discuss the tradeoffs between usability, security and privacy that are inherent to PAPTS schemes constructed as above.

12.4.1 Trust issues

Recall, from chapter 9, that one of the properties that a sound anonymous credential system must satisfy is that of *pseudonym owner protection*. According to this property, no entity, including any coalition of SPs and the proxy, should be able to successfully show a credential using a pseudonym that was not established by the entity to which the credential was issued.

Thus, in the protocol of section 12.3 above, the AS will not be able to show any credential under p_A or p_B (steps 4 and 5).

This observation illustrates a remarkable property of a PAPTS scheme constructed in the above way, namely that a dishonest or compromised proxy operator cannot impersonate users to SPs. In other words, the proxy does not need to be trusted not to impersonate users; it only needs be trusted to carry out user authentication properly. This contrasts with existing proxy-based SSO systems, where the proxy can typically impersonate users to all SPs in the system.

12.4.2 Security versus usability

The PAPTS schemes described in this chapter do not necessarily require more manual interaction from the user than any other proxy-based true SSO scheme. Manual interaction with the user is only required for the initial authentication to the AS (step 2); the subsequent issuing and showing of credentials could be handled by the user's software transparently and automatically. This software may, of course, need to have access to additional data, some of which may be secret. If cross-platform mobility is an issue, care has to be taken so that these secrets can follow the user. Ideally, the pseudonym system-related secrets could be stored together with the secrets required for the user authentication mechanisms (step 2); in this way, cross-platform mobility can be supported whenever the authentication context supports it.

The protocols that are executed in a PAPTS system of the above type are more complex and typically require more bandwidth and computation that those of traditional systems. The added overhead might be especially significant when the scheme is augmented with an anonymous user-to-SP channel, for example a MIX cascade [45]. This overhead would potentially adversely affect the system's usability; however, the overhead is a constant factor that could probably be reduced to a satisfactory level by making available additional bandwidth and computational power. We believe that, in certain scenarios, the benefits offered by the scheme are worth this additional cost.

In terms of tradeoffs between usability and security, the main tradeoff comes from the fact already mentioned in section 2.2 that a single authentication credential enables access to

more that one service. It is therefore of great importance for this credential to be maintained in a secure fashion, within the scope of the corresponding authentication context. Of course, in order for an adversary to impersonate the user to many SPs, it is not sufficient to compromise the user's single authentication credential; he also needs access to the user's pseudonym system-specific secrets. Since typically there is one such secret per established pseudonym, knowledge of multiple secrets is required to enable the adversary to gain access to more than one SP. However, if the above suggestion to store pseudonym-system related secrets together with those required for user authentication is implemented, it may be likely that compromising one of the user's secrets is essentially as simple as compromising them all. In this case, the usability/security tradeoffs of a PAPTS scheme are very similar to those applying to any SSO scheme.

12.4.3 Privacy versus security

As discussed in the previous chapters, it is desirable for the credential type t to encode the minimum amount of information necessary. The minimum amount of information that a SP is likely to require is a unique identifier for the authentication context c in use, and a timestamp τ that indicates the instant in time when the authentication (step 1 in section 12.3) took place; we henceforth assume that no other information is encoded into t. We also assume that the proxy and all SPs in the system maintain loosely synchronised clocks.

Each ordered pair (c, τ) represents a separate credential type. Thus, credentials that encode a different combination of context and time do not belong to the same anonymity set. In other words, users that were authenticated under a different context c, or at in a different time period (as encoded by τ), are distinguishable. Furthermore, as the user has to show the credential (in steps 4 and 5) before it has expired, the timestamp τ enables certain timing attacks, including those described in chapter 8, to be launched against the system. The objective of these attacks is to link the user's pseudonyms. As shown in that chapter, it is desirable for the accuracy of τ to be as coarse as possible in order to reduce the exposure to such attacks. From a security perspective, on the other hand, the accuracy of timestamps should be as fine-grained as possible. These contradicting requirements constitute a tradeoff that is perhaps the most fundamental in the construction of a PAPTS scheme. The choice for the accuracy of τ is therefore a sensitive issue.

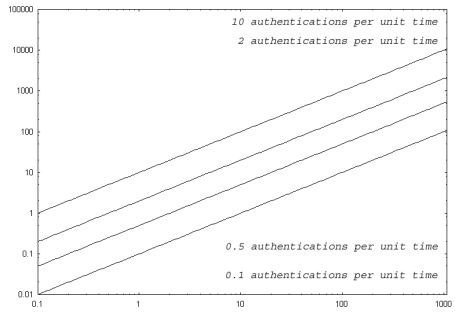

Figure 12.1: Average number of credentials issued per unit time (vertical axis) vs. accuracy of timestamp τ (horizontal axis).

Figure 12.1 shows how the accuracy of the timestamp τ affects anonymity. The vertical axis depicts the average number of pseudonyms on which a credential carrying a particular timestamp τ is issued, and the horizontal axis represents the length (in unit time) of periods in which time is divided by τ. Reading from top to bottom, the four lines represent the cases where, on average, 10, 2, 0.5 and 0.1 credentials are issued per unit time. Clearly, the relationship is linear. If, for example, 1 authentication occurs every 2 minutes (on distinct pseudonyms), a timestamp accuracy of 30 minutes gives an average anonymity set size of 15.

However, this measure does not adequately describe the anonymity of the system; while, on average, transactions may be hidden in relatively large anonymity sets, a substantial number of individual transactions may be exposed. Other statistical measures (such as standard deviation, skewness, etc.) of the authentication event distribution per unit time should be taken into account. By modeling incoming authentication requests as a Poisson process with arrival rate λ authentications per unit time, we can use the formula

$$P(n) = \frac{\lambda^n e^{-\lambda}}{n!}$$

in order to derive the probability $P(n)$ of n authentications occurring per unit time. The Poisson distribution has been used in the field of traffic engineering as a measure of incoming requests to a server per unit time [86], as well as in the context of anonymity systems [183]. We believe that it is reasonable to choose this distribution in order to model the process at hand.

Using this technique we can study the statistical behaviour of the system and how a particular selection of parameters affects the size of the anonymity set in which authentication

events are hidden. For example, we can calculate the percentage of time periods (as encoded by τ) in which at least 5 authentications occur, given an average of, say, $\lambda = 10$, as follows.

$$1 - \sum_{n=0}^{4} P(n) \approx 1 - 4.539 \times 10^{-5} - 4.539 \times 10^{-4} - 2.699 \times 10^{-3} - 7.566 \times 10^{-3} - 1.891 \times 10^{-2} \approx 97\%$$

This result essentially states that, if 10 credentials are issued (on average) with a common timestamp τ, 97% of these credentials will share a common τ with at least four other credentials; the remaining 3% will share their type with three other credentials or less.

Similarly, we can calculate the required accuracy of the timestamp τ in order to achieve a given confidence level in the minimum number of pseudonyms on which a credential is issued in each period. For example, in order for, say, 95% of credentials to be issued in a period in which at least 10 credentials are issued in total, the average number of incoming authentications per unit time has to be $\lambda = 3.5$[12.1]. In other words, in a system with, say, 5 incoming authentications per minute on average, the accuracy of τ needs to be 3.1 minutes (186 seconds) in order for 95% of credentials to be issued in a period in which at least 10 credentials are issued in total.

The above statistical analysis helps us estimate the number of credentials that are issued (or are not issued) within periods in which sufficiently many credentials are issued in total. Two issues arise here. Firstly, the definition of 'sufficiently many' is likely to be different from user to user. Secondly, even with a, say, 99% chance of being authenticated in a period in which sufficiently many other users do the same, a user might not wish to have even a 1% chance of his anonymity being compromised. For users that require stronger guarantees than those offered by probabilistic arguments, the scheme described in chapter 10 seems to be particularly appropriate.

The peer-to-peer synchronisation scheme of chapter 10 addresses both the above issues. Firstly, it allows the user to specify the desired minimum number of users that should gather in order to synchronise their acquisition of a credential (i.e. be authenticated by the same mechanism). Secondly, in the absence of attacks not addressed by the scheme, the number of users actually obtaining the credential is always at least the specified minimum,

[12.1] A more precise approximation for the percentage of such credentials, given $\lambda = 15.5$, is 94.8%. This value has been derived from the table of cumulative Poisson probabilities `ugrad.stat.ubc.ca/ stat241/tables/Poisson` (row $x = 9$, column $\alpha = 15.5$).

100% of the time.

12.4.4 Usability versus privacy

In this section we discuss the tradeoffs between usability and privacy that are inherent in a PAPTS scheme constructed as described in section 12.3.

Consider a setting where the AS offers two authentication contexts, denoted c_1 and c_2. Suppose that context c_1 is stronger than c_2, meaning that SPs that accept credentials of type (c_2, τ) also accept credentials of type (c_1, τ) (but not vice versa). Consider a user that has obtained a credential of type $t_1 = (c_1, \tau_1)$ in order to log into an SP that requires a credential that encodes this stronger authentication context. Next, the user wishes to log into an SP that requires only context c_2, and therefore also accepts the above credential. The question that arises in this situation is the following. Assuming that the credential (c_1, τ_1) has not expired, is it a better strategy for the user to (a) obtain a credential (c_2, τ_2) by first authenticating himself to the AS under context c_2, or (b) reuse the credential (c_1, τ_1)? Obviously, from a usability perspective, it is preferable to reuse the existing credential, as obtaining a new one requires the user to re-authenticate himself under context c_2, and might therefore require manual interaction.

We now attempt to answer this question from a privacy perspective. If there were more credentials issued that encode context c_1 in time period τ_1 than credentials encoding context c_2 in time period τ_2, then it is also preferable to reuse the same credential from a privacy perspective. This is because linking the credential showing to its issuing would be harder in this case. Of course, if more credentials of type (c_2, τ_2) were issued than credentials of type (c_1, τ_1), then the reverse applies.

Without some system that gives relevant feedback to the user, such as the peer-to-peer scheme of chapter 10, there is no way for him to know how many similar credentials are issued in any given time period, and, thus the user cannot decide which strategy is better from a privacy perspective. So, one solution is to use a scheme that provides appropriate feedback to the user and then reuse the same credential. However, this analysis only considers the issue from the point of view of a single user. As we have seen in the previous chapters in this part of this thesis, from a global privacy perspective it is preferable for as many credentials

to be issued as possible. Thus, if the user follows strategy (b) above, although he himself may not be better off in terms of privacy (i.e. transaction unlinkability), he contributes to the privacy of other users in the system. Thus, if all users follow that strategy, i.e. obtain a credential encoding an authentication context of minimally accepted strength whenever possible, every user's privacy would be enhanced.

Note that following this strategy does not mean loss of the SSO functionality; a user still reuses the same credentials with SPs that require the same authentication context. However, following that strategy is likely to impair usability without buying the individual more privacy. It is a challenge to incorporate mechanisms into this system that will motivate users to act selflessly for the 'overall good'. This challenge also extends to other privacy-protecting systems.

12.5 Summary

This chapter described a method for constructing PAPTS schemes using a series of user authentication mechanisms and an anonymous credential system. Such a scheme enables users to be authenticated by a centralised proxy, and then obtain services at a number of SPs. A PAPTS system constructed in this way has two important properties, namely that (a) no coalition of SPs and the proxy can learn which pseudonyms belong to which users, and (b) the proxy cannot impersonate users at SPs. In particular, it is only trusted to perform authentication properly. It was pointed out that one of the most sensitive issues in this setting is the choice for the accuracy of the timestamp that indicates the time of authentication at the proxy. A weakness is the fact that users do not get feedback on the level of unlinkability their pseudonyms actually enjoy in a working system. The scheme proposed in chapter 10 could be used to address this issue.

Part III

Conclusions

Chapter 13

Conclusions and directions for further research

Contents

This chapter presents the overall conclusions of this thesis and gives some directions for further research.

13.1 Conclusions

This thesis contains a variety of contributions. In the first part, four novel and pragmatic interdomain user authentication schemes were proposed and their properties were discussed. Three of these schemes were based on existing deployed infrastructures, namely GSM/UMTS telephony, EMV credit/debit smartcards, and computing platforms conformant to the TCG specification, respectively. The overall level of security of these schemes is likely to be higher than that provided by many existing schemes because of the added protection offered by the tamper-resistance of the hardware used in these schemes. Furthermore (with the exception of the scheme described in chapter 6), key management practices for the underlying security infrastructures are well defined and have been in place for some time. This is likely to contribute to smooth and secure operations in the SSO context as well.

The schemes demonstrate that it is possible to reuse existing security infrastructures in contexts other than that in which they were originally deployed. Of course, care has to be taken to separate application domains in order to avoid security vulnerabilities and potential

attacks that apply in one domain extending to another.

A further important advantage gained by reusing existing infrastructure in a different context is that the schemes of chapters 4, 5, and 6 are potentially cheap and straightforward to deploy, and hence are fundamentally pragmatic solutions. However, the systems also inherit properties of the underlying infrastructure that may be undesirable in the new environment. For example, the schemes based on GSM/UMTS and EMV, described in chapters 4 and 5 respectively, enable cooperating SPs to trivially correlate the identifiers of any given user. This 'feature' is inherited from the underlying security infrastructure, and may be undesirable in scenarios where privacy protection plays an important role. The situation with respect to the exploitation of TCG-conformant platforms is a little different, as the TCG-based infrastructure was always intended to be 'general purpose' in nature; in particular, it provides sophisticated privacy functionality, especially in the case of platforms conforming to version 1.2 of the TCG specifications.

The fourth SSO scheme of part I, described in chapter 7, does not rely on a existing deployed infrastructure. It has the unique feature that no long-term secrets are disclosed to the user's local machine, while enabling the user to log into SPs that use arbitrary user authentication methods. In other words, it combines two services that previously did not exist in a single SSO system: it both provides one-time authentication, and is transparent to SPs. The scheme was also implemented in a 'real-world' setting as a web proxy and was successfully tested with a variety of browsers. The scheme is useful in scenarios where a roaming user needs to use untrusted machines or network access devices.

Part II of the thesis examined a type of cryptosystem called an 'anonymous credential' or a 'pseudonym' system. In chapter 8 certain timing attacks that apply to anonymous credential system were described. The aim of these attacks is to link otherwise unlinkable identifiers of a given user using timing information. A complexity theoretic adversarial model was then formulated in chapter 9, and applicable security and privacy notions were defined. In chapter 10 a peer-to-peer scheme was proposed that offers a degree of protection against some of the attacks described in chapter 8, and additionally provides real-time feedback to users of anonymous credential systems.

The idea of using an anonymous credential system in order to build what we call a 'privacy-aware' SSO scheme is also explored in part II of the thesis. In particular, two methods of

combining a pseudonym system with user authentication schemes are discussed in chapter 12. The first method, although more complicated than the second, is flexible in the sense that any user authentication mechanism is supported. The second method, although simpler, is less flexible in the sense that the only supported user authentication mechanisms are those that follow the pseudonym system protocols.

More generally, we believe that there is a lack of available technology that protect user privacy in the electronic world; moreover, the technology that does exist fails to meet all the potential user privacy requirements. Thus, the construction, implementation and analysis of schemes to protect user privacy remains a challenge that is not only open, but becomes more important as the number of available services grows. Moreover, emerging and novel network architectures and the associated applications and services pose new risks to user privacy and, as such, require innovative solutions. It is therefore important to raise awareness and develop appropriate privacy protecting technologies. It is hoped that this thesis contributes towards such an effort.

13.2 Directions for further research

Throughout the thesis a number of directions for further research have been identified. This section summarises what we believe to be the most important and interesting ones. We divide potential further research directions into two categories, namely practical and theoretical. On the practical side, we believe that the following directions are of interest.

- Implementation of real-world privacy-aware SSO schemes, as described in chapter 12, possibly with the add-on scheme described in chapter 10 or using some other anonymous user-to-SP channel.
- Integration of SSO and privacy-protecting identity management systems.
- Identification of applications in the area of ubiquitous computing (e.g. sensor networks) for the scheme described in chapter 7.
- Integration of an anonymous credential system (e.g. that of Camenisch and van Herreweghen [40]) and the peer-to-peer scheme described in chapter 10.

On the theoretical side, open questions arise in connection with the security model presented in chapter 9. As mentioned in section 9.3, it is a challenge to construct cryptosystems that satisfy the definitions of the model. It is also interesting to discover whether or not existing schemes meet the definitions. Finally, it would also be desirable to refine the model so that it captures additional properties such as anonymity revocation and the use of credentials that can only be shown a predetermined number of times.

Bibliography

[1] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

[2] B. Aboba, L. Blunk, J. Vollbrecht, and J. Carlson. *RFC 3748: Extensible Authentication Protocol (EAP)*, 2004.

[3] G.-J. Ahn, D. Shin, and S.-P. Hong. Information assurance in federated identity management: Experimentations and issues. In X. Zhou, S. Y. W. Su, M. P. Papazoglou, M. E. Orlowska, and K. G. Jeffery, editors, *Web Information Systems — WISE 2004, 5th International Conference on Web Information Systems Engineering, Brisbane, Australia, November 22-24, 2004, Proceedings*, number 3306 in Lecture Notes in Computer Science, pages 79–90. Springer Verlag, Berlin, November 2004.

[4] M. A. Al-Meaither and C. J. Mitchell. A secure GSM-based Murabaha transaction. In *Proceedings of the 1st International Conference on Information & Communication Technologies from Theory to Applications (ICTTA)*, pages 77–78. IEEE Press, April 2004.

[5] American National Standards Institute. *ANSI 9.84-2003: Biometric Information Management and Security for the Financial Services Industry*, 2003.

[6] T. Aura and P. Nikander. Stateless connections. In Y. Han, T. Okamoto, and S. Quing, editors, *ICICS '97: Proceedings of the First International Conference on Information and Communication Security*, volume 1334 of *Lecture Notes in Computer Science*, pages 87–97, London, UK, 1997. Springer-Verlag.

[7] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In I. S. Moskowitz, editor, *Information Hiding, 4th International Workshop, IHW 2001*, volume 2137 of *Lecture Notes in Computer Science*, pages 245–257. Springer Verlag, Berlin, 2001.

[8] M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *IEEE Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.

[9] B. Balacheff, L. Chen, S. Pearson, D. Plaquin, and G. Proudler. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice-Hall, 2003.

[10] G. Barish and K. Obraczka. World wide web caching: Trends and techniques. *IEEE Communications Magazine*, 38(5):178–185, May 2000.

[11] M. Bellare, R. Canetti, and H. Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proceedings of the 37th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 514–523. IEEE, 1996.

[12] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the 30th Annual Symposium on the Theory of Computing*, pages 419–428. ACM, 1998.

[13] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 394–403. IEEE Computer Society, 1997.

[14] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology – CRYPTO 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer-Verlag, 1998.

[15] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *Advances in Cryptology — Asiacrypt 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545. Springer-Verlag, Berlin, 2000.

[16] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73. ACM, 1993.

[17] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Advances in Cryptology – CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, Berlin, 1994.

[18] M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing STOC*, pages 57–66. ACM, 1995.

[19] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In A. Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

[20] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. In *USENIX Conference Proceedings*, pages 253–267, Dallas, TX, Winter 1991. USENIX.

[21] T. Berners-Lee, L. Masinter, and M. M. (editors). *Uniform Resource Locators*, 2004.

[22] O. Berthold and M. Köhntopp. Identity management based on P3P. In H. Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, July 2000*, number 2009 in Lecture Notes in Computer Science, pages 141–160. Springer-Verlag, Berlin, 2001.

[23] A. Biryukov, J. Lano, and B. Preneel. Cryptanalysis of the alleged SecurID hash function. Cryptology ePrint Archive, Report 2003/162, 2003. `http://eprint.iacr.org/`.

[24] S. Blake-Wilson and A. Menezes. Authenticated Diffie-Hellman key agreement protocols. In S. E. Tavares and H. Meijer, editors, *Selected Areas in Cryptography '98, SAC'98, Kingston, Ontario, Canada, August 17-18, 1998, Proceedings*, volume 1556 of *Lecture Notes in Computer Science*, pages 339–361. Springer Verlag, Berlin, 1999.

[25] C. W. Blanchard. Wireless security. In R. Temple and J. Regnault, editors, *Internet and wireless security*, chapter 8, pages 147–162. IEE, 2002.

[26] A. Boldyreva. Efficient threshold signature, multisignature and blind signature schemes based on the gap-Diffie-Hellman-group signature scheme. In Y. Desmedt, editor, *International Workshop on Practice and Theory in Public Key Cryptography – PKC 2003*, volume 2567 of *Lecture Notes in Computer Science*, pages 31–46. Springer-Verlag, 2003.

[27] C. Boyd. A framework for design of key establishment protocols. In J. Pieprzyk and J. Seberry, editors, *Australasian Conference on Information Security and Privacy*,

volume 1172 of *Lecture Notes in Computer Science*, pages 146–157. Springer Verlag, Berlin, 1996.

[28] C. Boyd and W. Mao. On a limitation of BAN logic. In T. Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 240–247. Springer-Verlag, Berlin, 1994.

[29] C. Boyd and A. Mathuria. Key establishment protocols for secure mobile communications: A selective survey. In C. Boyd and E. Dawson, editors, *Information Security and Privacy: Third Australasian Conference, ACISP'98, Brisbane, Australia, July 1998. Proceedings*, volume 1438 of *Lecture Notes in Computer Science*, pages 344–355. Springer Verlag, Berlin, 1998.

[30] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment.* Springer Verlag, 2003.

[31] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates — Building in Privacy.* The MIT Press, Cambridge, Massachusetts, 2000.

[32] D. Branstad. Security aspects of computer networks. In *AIAA Computer Network Systems Conference*, Huntsville, Alabama, April 1973. AIAA Paper No. 73-427.

[33] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 132–145, New York, NY, USA, 2004. ACM Press.

[34] G. Brown. The use of hardware tokens for identity management. *Information Security Technical Report*, 9(1):22–25, January–March 2004.

[35] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, Digital Systems Research Center, February 1989.

[36] J. Camenisch. Better privacy for trusted computing platforms: (extended abstract). In P. Samarati, D. Gollmann, and R. Molva, editors, *Computer Security - ESORICS 2004: 9th European Symposium on Research in Computer Security, Sophia Antipolis, France, September 13 - 15, 2004. Proceedings*, volume 3193 of *Lecture Notes in Computer Science*, pages 73–88, 2004.

[37] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *Advances in*

Cryptology — EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceedings, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer Verlag, Berlin, 2001.

[38] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *Advances in Cryptology — CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 61–76. Springer Verlag, Berlin, 2002.

[39] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *Proceedings of the 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19 — CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 56–72. Springer-Verlag, Berlin, 2004.

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

[41] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE Computer Society, 2001.

[42] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing*, pages 209–218. ACM, 1993.

[43] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In B. Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474. Springer-Verlag, 2001.

[44] R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 2002.

[45] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[46] D. Chaum. Blind signatures for untraceable payments. In R. Rivest, A. Sherman, and D. Chaum, editors, *Advances in Cryptology – CRYPTO 82*, pages 199–203. Plenum Press, 1983.

[47] D. Chaum. Blind signature system. In D. Chaum, editor, *Advances in Cryptology – CRYPTO 83*, page 153. Plenum Press, 1984.

[48] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, October 1985.

[49] D. Chaum. Privacy protected payments: Unconditional payer and/or payee untraceability. In D. Chaum and I. Schaumueller-Bichl, editors, *SMART CARD 2000*, pages 69–93. Elsevier Science Publishers B.V., 1989.

[50] D. Chaum. Showing credentials without identification: Transferring signatures between unconditionally unlinkable pseudonyms. In J. Seberry and J. Pieprzyk, editors, *Advances in Cryptology – AUSCRYPT 90*, volume 453 of *Lecture Notes in Computer Science*, pages 246–264. Springer-Verlag, Berlin, 1990.

[51] D. Chaum. *One-show blind signature systems.* U.S. Patent ser. no. 4,987,593. Filed April 1990. Continuation of abandoned application Ser. No. 07/168,802, filed March 1988, January 1991.

[52] D. Chaum. *Unpredictable blind signature systems.* U.S. Patent serial number 4,991,210. Filed May 1989., February 1991.

[53] D. Chaum. Achieving electronic privacy. *Scientific American*, 267(2):96–101, August 1992.

[54] D. Chaum and J.-H. Evertse. A secure and privacy-protecting protocol for transmitting personal information between organizations. In A. M. Odlyzko, editor, *Advances in Cryptology — CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, number 263 in Lecture Notes in Computer Science, pages 118–168. Springer Verlag, Berlin, 1987.

[55] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In S. Goldwasser, editor, *Advances in Cryptology – CRYPTO 88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer-Verlag, Berlin, 1988.

[56] L. Chen. Access with pseudonyms. In E. Dawson and J. D. Golic, editors, *Cryptography: Policy and Algorithms, International Conference, Brisbane, Queensland, Australia, July 3-5, 1995, Proceedings*, number 1029 in Lecture Notes in in Computer Science, pages 232–243. Springer Verlag, Berlin, 1995.

[57] J. Claessens, B. Preneel, and J. Vandewalle. Combining World Wide Web and wireless security. *Informatica*, 26(2):123–132, 2002.

[58] S. Clauß and M. Köhntopp. Identity management and its support of multilateral security. *Comput. Networks*, 37(2):205–219, 2001.

[59] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips. *Universal Serial Bus Specification*, 2nd edition, April 2000.

[60] Computer Security Center of the Department of Defense, Meade, Fort George G., Maryland 20755. *Department of Defense Password Management Guideline*, April 1985. CSC-STD-002-85.

[61] S. Contini and Y. L. Yin. Improved cryptanalysis of SecurID. Cryptology ePrint Archive, Report 2003/205, 2003.

[62] B. P. Cosell, P. R. Johnson, J. H. Malman, R. E. Schantz, J. Sussman, R. H. Thomas, and D. C. Walden. An operational system for computer resource sharing. In *SOSP '75: Proceedings of the fifth ACM symposium on operating systems principles*, pages 75–81. ACM Press, 1975.

[63] I. Damgård. Payment systems and credential mechanisms with provable security against abuse by individuals. In S. Goldwasser, editor, *Advances in Cryptology — CRYPTO '88: Proceedings*, number 403 in Lecture Notes in Computer Science, pages 328–335. Springer Verlag, 1990.

[64] D. W. Davies and W. L. Price. *Security for computer networks: an introduction to data security in teleprocessing and electronic funds transfer*. John Wiley & Sons, Inc., 2nd edition, 1989.

[65] J. De Clercq. Single sign-on architectures. In G. I. Davida, Y. Frankel, and O. Rees, editors, *Infrastructure Security, International Conference, InfraSec 2002 Bristol, UK, October 1-3, 2002, Proceedings*, volume 2437 of *Lecture Notes in Computer Science*, pages 40–58. Springer Verlag, 2002.

[66] Y. Demchenko. Virtual organisations in computer grids and identity management. *Information Security Technical Report*, 9(1):59–76, January–March 2004.

[67] A. Dent and C. Mitchell. *User's Guide to Cryptography and Standards*. Artech House, 2005.

[68] C. Díaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. In R. Dingledine and P. F. Syverson, editors, *Proceedings of Privacy Enhancing Technologies, 2nd International Workshop, PET 2002*, number 2482 in Lecture Notes in Computer Science, pages 54–68. Springer-Verlag, Berlin, 2002.

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

[70] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1983.

[71] J. R. Douceur. The Sybil attack. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *Peer-to-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer-Verlag, Berlin, 2002.

[72] J. Edwards. Single sign-on technology streamlines network access. *Software Magazine*, 13(17):35–42, 1993.

[73] Electronic Industries Alliance. *EIA232E: Interface between Data Terminal Equipment and Data Circuit Terminating Equipment employing serial binary data interchange*, 1991.

[74] J. H. Ellis. The possibility of secure non-secret digital encryption. Report, CESG, January 1970.

[75] EMV. *EMV2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 — Book 1: Application Independent ICC to Terminal Interface Requirements*, December 2000.

[76] EMV. *EMV2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 — Book 2: Security and Key Management*, December 2000.

[77] EMV. *EMV2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 — Book 3: Application Specification*, December 2000.

[78] EMV. *EMV2000 Integrated Circuit Card Specification for Payment Systems Version 4.0 — Book 4: Cardholder, Attendant and Acquirer Interface Requirements*, December 2000.

[79] European Telecommunications Standards Institution (ETSI). *Digital cellular telecommunications system (Phase 2+); Security aspects (GSM 02.09 version 8.0.1)*, June 2001.

[80] European Telecommunications Standards Institution (ETSI). *Digital cellular telecommunications system (Phase 2+); Security related network functions (GSM 03.20 version 8.1.0)*, July 2001.

[81] D. Flanagan. *Java in a Nutshell*. O'Reilly, 3rd edition, November 1999.

[82] W. Ford and M. Baum. *Secure Electronic Commerce*. Prentice Hall, 1996.

[83] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*. Internet Engineering Task Force, June 1999.

[84] S. Galbraith and W. Mao. Invisibility and anonymity of undeniable and confirmer signatures. In M. Joye, editor, *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2003.

[85] R. Ganesan. Yaksha: augmenting Kerberos with public key cryptography. In *SNDSS '95: Proceedings of the 1995 Symposium on Network and Distributed System Security (SNDSS'95)*, pages 132–143, Washington, DC, USA, 1995. IEEE Computer Society.

[86] M. Ghanbari, C. Hughes, M. Sinclair, and J. Eade. *Principles of Performance Engineering for Telecommunication and Information Systems*. Institution of Electrical Engineers, 1997.

[87] O. Goldreich. Randomness, interactive proofs, and zero-knowledge – a survey. In R. Herken, editor, *The Universal Turing Machine: A Half Century Survey*, pages 377–405. Oxford University Press, 1988.

[88] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of the ACM*, 38(3):690–728, 1991.

[89] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):84–88, January 1999.

[90] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.

[91] S. Gritzalis, D. Spinellis, and P. Georgiadis. Security protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification. *Computer Communications*, 22(8):697–709, May 1999.

[92] T. Gross. Security analysis of the SAML single sign-on browser/artifact profile. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 298–307. IEEE Press, December 2003.

[93] T. Gross and B. Pfitzmann. Proving a WS-federation passive requestor profile. In *ACM Secure Web Services Workshop*. ACM Press, 2004. to appear.

[94] M. F. Grubb and R. Carter. Single sign-on and the system administrator. In *Proceedings of the Twelfth Systems Administration Conference (LISA 98)*. Usenix, 1998.

[95] M. Hansen, P. Berlich, J. Camenisch, S. Clau, A. Pfitzmann, and M. Waidner. Privacy-enhancing identity management. *Information Security Technical Report*, 9(1):35–44, January–March 2004.

[96] S. M. Hansen, J. Skriver, and H. R. Nielson. Using static analysis to validate the saml single sign-on protocol. In *WITS '05: Proceedings of the 2005 workshop on Issues in the theory of security*, pages 27–40, New York, NY, USA, 2005. ACM Press.

[97] IEEE. *IEEE 1284.1 Standard for Information TechnologyTransport Independent Printer/System Interface (TIP/SI)*, 1997.

[98] IEEE. *Standard 802.11b-1999/Cor 1-2001(Corrigendum to IEEE Std 802.11b-1999)*, 1999-2001.

[99] Internet Engineering Task Force. *RFC 1510: The Kerberos Network Authentication Service (V5)*, September 1993.

[100] Internet Engineering Task Force. *RFC 2898: PKCS #5: Password-Based Cryptography Specification Version 2.0*, September 2000.

[101] Internet Engineering Task Force. *RFC 2821: Simple Mail Transfer Protocol*, April 2001.

[102] Internet Engineering Taskforce. *Extensible Authentication Protocol Method for GSM Subscriber Identity Modules (EAP-SIM)*, December 2004. work in progress.

[103] L. Ishitani, V. Almeida, and W. M. Jr. Masks: Bringing anonymity and personalization together. *IEEE Security and Privacy*, 1(3):18–23, May–June 2003.

[104] ITU-T Recommendation X.509. *Information technology — Open Systems Interconnection — The Directory: Public-key and attribute certificate frameworks*, 2000.

[105] B. Ives, K. R. Walsh, and H. Schneider. The domino effect of password reuse. *Communications of the ACM*, 47(4):75–78, April 2004.

[106] U. Jendricke and D. G. tom Markotten. Usability meets security — the identity-manager as your personal security assistant for the internet. In *Proceedings of the 16th Annual Computer Security Applications Conference*, pages 344–355. IEEE Computer Society, 2000.

[107] J. Jeong, D. Shin, D. Shin, and K. Moon. Java-based single sign-on library supporting SAML for distributed web services. In J. X. Yu, X. Lin, H. Lu, and Y. Zhang, editors, *Advanced Web Technologies and Applications, 6th Asia-Pacific Web Conference, APWeb 2004, Hangzhou, China, April 14-17, 2004*, volume 3007 of *Lecture Notes in Computer Science*, pages 891–894. Springer Verlag, Berlin, 2004.

[108] A. Jøsang and M. A. Patton. User interface requirements for authentication of communication. In *CRPITS '18: Proceedings of the Fourth Australian user interface conference on User interfaces 2003*, pages 75–80, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[109] W. K. Josephson, E. G. Sirer, and F. B. Schneider. Peer-to-peer authentication with a distributed single sign-on service. In G. M. Voelker and S. Shenker, editors, *Peer-to-Peer Systems III, Third International Workshop, IPTPS 2004, La Jolla, CA, USA, February 26-27, 2004, Revised Selected Papers*, volume 3279 of *Lecture Notes in Computer Science*, pages 250–258. Springer, 2005.

[110] A. Juels, M. Luby, and R. Ostrovsky. Security of blind digital signatures. In B. S. Kaliski, editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164, London, UK, 1997. Springer-Verlag.

[111] J. Katz and M. Yung. Complete characterization of security notions for probabilistic private-key encryption. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on theory of computing*, pages 245–254. ACM Press, 2000.

[112] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World*. Prentice Hall, 2nd edition, 2002.

[113] R. A. Kemmerer, C. Meadows, and J. K. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[114] S. T. Kent. Encryption-based protection protocols for interactive user-computer communication. Laboratory for Computer Science Technical Report 162, Massachusetts Institute of Technology, May 1976.

[115] S. T. Kent. Encryption-based protection for interactive user/computer communication. In *Proceedings of the fifth symposium on data communications*, pages 5.7–5.13. ACM Press, 1977.

[116] R. Khare and S. Lawrence. *Upgrading to TLS Within HTTP/1.1*, 2000.

[117] V. Khu-Smith and C. Mitchell. Using GSM to enhance e-commerce security. In *Proceedings of the Second ACM International Workshop on Mobile Commerce (WMC '02)*, pages 75–81, New York, 2002. ACM Press.

[118] N. Koblitz and A. Menezes. Another look at "provable security". Cryptology ePrint Archive, Report 2004/152, 2004. `http://eprint.iacr.org/`.

[119] J. Kohl, B. Neuman, and T. Ts'o. The evolution of the Kerberos authentication service. In *Distributed Open Systems*, pages 78–94. IEEE Computer Society Press, 1994.

[120] D. P. Kormann and A. D. Rubin. Risks of the Passport single signon protocol. In *Proceedings of the 9th international World Wide Web conference on computer networks : the international journal of computer and telecommunications networking*, pages 51–58, Amsterdam, The Netherlands, The Netherlands, 2000. North-Holland Publishing Co.

[121] H. Krawczyk. Simple forward-secure signatures from any signature scheme. In *CCS '00: Proceedings of the 7th ACM conference on computer and communications security*, pages 108–115, New York, NY, USA, 2000. ACM Press.

[122] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Des. Codes Cryptography*, 28(2):119–134, 2003.

[123] C.-C. Lee, W.-P. Yang, and M.-S. Hwang. Untraceable blind signature schemes based on discrete logarithm problem. *Fundam. Inf.*, 55(3-4):307–320, 2003.

[124] J.-Y. Lee, J. H. Cheon, and S. Kim. An analysis of proxy signatures: Is a secure channel necessary? In M. Joye, editor, *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2003.

[125] B. N. Levine, M. Reiter, C. Wang, and M. Wright. Stopping timing attacks in low-latency mix-based systems. In A. Juels, editor, *Proceedings of Financial Cryptography, 8th International Conference, FC 2004, Key West, FL, USA, February 9-12*, volume 3110 of *Lecture Notes in Computer Science*. Springer, Berlin, 2004.

[126] B. Li, S. Ge, T. Wo, and D. Ma. Research and implementation of single sign-on mechanism for ASP pattern. In H. Jin, Y. Pan, N. Xiao, and J. Sun, editors, *Grid and Cooperative Computing - GCC 2004: Third International Conference, Wuhan, China, October 21-24, 2004. Proceedings*, volume 3251 of *Lecture Notes in Computer Science*, pages 161–166. Springer, 2004.

[127] Liberty Alliance. *Identity Systems and Liberty Specification, version 1.1, Interoperability*, January 2003.

[128] Liberty Alliance. *Liberty Architecture Glossary v.1.2-04*, April 2003.

[129] Liberty Alliance. *Liberty Authentication Context Specification v.1.2-05*, April 2003.

[130] Liberty Alliance. *Liberty ID-FF Architecture Overview v.1.2-03*, April 2003.

[131] Liberty Alliance. *Liberty ID-FF Bindings and Profiles Specification v.1.2-08*, April 2003.

[132] Liberty Alliance. *Liberty ID-FF Implementation Guidelines v.1.2-02*, April 2003.

[133] Liberty Alliance. *Liberty ID-FF Protocols and Schema Specification v.1.2-08*, April 2003.

[134] M. Linden and I. Vilpola. An empirical study on the usability of logout in a single sign-on system. In R. H. Deng, F. Bao, H. Pang, and J. Zhou, editors, *Proceedings of the First Information Security Practice and Experience Conference (ISPEC 2005)*, volume 3439 of *Lecture Notes in Computer Science*, pages 243–254. Springer Verlag, Berlin, 2005.

[135] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[136] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.

[137] A. Lysyanskaya. *Signature schemes and applications to cryptographic protocol design.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 2002.

[138] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In H. M. Heys and C. M. Adams, editors, *Selected Areas in Cryptography, 6th Annual International Workshop, SAC'99, Kingston, Ontario, Canada, August 9-10, 1999, Proceedings*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer Verlag, Berlin, 2000.

[139] W. Mao. *Modern Cryptography: Theory and Practice.* Prentice Hall PTR, 2003.

[140] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–36, 1992.

[141] A. Menezes, M. Qu, and S. Vanstone. Some new key agreement protocols providing mutual implicit authentications. *Proceedings of the 2nd Workshop on Selected Areas in Cryptography (SAC'95), Carleton University, Ottawa, Canada, May 1995*, pages 22–32, May 1995.

[142] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1997.

[143] R. C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294–299, 1978.

[144] Microsoft. *Microsoft .NET Passport Review Guide*, November 2002.

[145] S. Nanavati, M. Thieme, and R. Nanavati. *Biometrics: Identity Verification in a Networked World.* Wiley, March 2002.

[146] National Bureau of Standards, U.S. Department of Commerce, Washington D.C. *Federal Information Processing Standards Publication 46-3: Data Encryption Standard(DES)*, October 1999.

[147] National Institute of Standards and Technology. *Federal Information Processing Standards Publication 180-1: Secure Hash Standard*, April 1995.

[148] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[149] OASIS, http://www.oasis-open.org/committees/security/. *Security Services Technical Commitee Homepage.*

[150] OASIS. *Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)*, May 2002.

[151] R. Oppliger. Microsoft .NET passport and identity management. *Information Security Technical Report*, 9(1):26–34, January–March 2004.

[152] P. Pagliusi and C. J. Mitchell. PANA/GSM authentication for Internet access. In *Proceedings of SympoTIC '03, Joint IST Workshop on Mobile Future and Symposium on Trends in Communications*, pages 146–152. IEEE Press, October 2003.

[153] A. Pashalidis. A cautionary note on automatic proxy configuration. In M. Hamza, editor, *IASTED International Conference on Communication, Network, and Information Security, CNIS 2003, New York, USA, December 10-12, 2003, Proceedings*, pages 153–158. ACTA Press, December 2003.

[154] A. Pashalidis and C. Mitchell. A security model for anonymous credential systems. In S. J. Y. Deswarte, F. Cuppens and L. Wang, editors, *Information Security Management, Education and Privacy, Proceedings of the 3rd Working Conference on Privacy and Anonymity in Networked and Distributed Systems (I-NetSec'04)*, pages 183–199. Kluwer Academic Publishers, August 2004.

[155] A. Pashalidis and C. Mitchell. Using EMV cards for single sign-on. In S. K. Katsikas, S. Gritzalis, and J. Lopez, editors, *Public Key Infrastructure, First European PKI-Workshop: Research and Applications, EuroPKI 2004, Samos Island, Greece, June 25-26, 2004, Proceedings*, volume 3093 of *Lecture Notes in Computer Science*, pages 205–217. Springer Verlag, June 2004.

[156] A. Pashalidis and C. J. Mitchell. Single sign-on using trusted platforms. In C. Boyd and W. Mao, editors, *Information Security, 6th International Conference, ISC 2003, Bristol, UK, October 1-3, 2003, Proceedings*, volume 2851 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, October 2003.

[157] A. Pashalidis and C. J. Mitchell. A taxonomy of single sign-on systems. In R. Safavi-Naini and J. Seberry, editors, *Information Security and Privacy – 8th Australasian Conference, ACISP*, volume 2727 of *Lecture Notes in Computer Science*, pages 249–264. Springer Verlag, July 2003.

[158] A. Pashalidis and C. J. Mitchell. Using GSM/UMTS for single sign-on. In *Proceedings of SympoTIC '03, Joint IST Workshop on Mobile Future and Symposium on Trends in Communications, Bratislava, Slovakia*, pages 138–145. IEEE Press, October 2003.

[159] A. Pashalidis and C. J. Mitchell. Impostor: A single sign-on system for use from untrusted devices. In *Proceedings of the IEEE Globecom Conference, Dallas, Texas, USA, November 29 – December 3*. IEEE Press, 2004.

[160] A. Pashalidis and C. J. Mitchell. Single sign-on using trusted platforms. In C. J. Mitchell, editor, *Trusted Computing*, chapter 6, pages 175–193. IEE Press, London, 2005.

[161] A. Pashalidis and C. J. Mitchell. Limits to anonymity when using credentials. In *Proceedings of the 12th International Workshop on Security Protocols, Cambridge, U.K.*, Lecture Notes in Computer Science. Springer Verlag, to appear.

[162] T. P. Pedersen and B. Pfitzmann. Fail-stop signatures. *SIAM J. Comput.*, 26(2):291–330, 1997.

[163] G. Persiano and I. Visconti. An efficient and usable multi-show non-transferable anonymous credential system. In A. Juels, editor, *Proceedings of the Eighth International Financial Cryptography Conference (FC '04)*, volume 3110 of *Lecture Notes in Computer Science*, pages 196–211, 2004.

[164] A. Pfitzmann and M. Köhntopp. Anonymity, unobservability, and pseudonymity - a proposal for terminology. In H. Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, July 2000*, number 2009 in Lecture Notes in Computer Science, pages 141–160. Springer-Verlag, Berlin, 2001.

[165] B. Pfitzmann. Privacy in enterprise identity federation — policies for Liberty 2 single sign on. *Information Security Technical Report*, 9(1):45–58, January–March 2004.

[166] B. Pfitzmann. Privacy in enterprise identity federation — policies for Liberty single signon. In *Proceeings: 3rd Workshop on Privacy Enhancing Technologies (PET 2003),*

Dresden, March 2003, Lecture Notes in Computer Science. Springer-Verlag, Berlin, to appear.

[167] B. Pfitzmann and M. Waidner. Privacy in browser-based attribute exchange. In S. Jajodia and P. Samarati, editors, *WPES '02: Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 52–62, New York, NY, USA, 2002. ACM Press.

[168] B. Pfitzmann and M. Waidner. Analysis of Liberty single-sign-on with enabled clients. *Internet Computing*, 7(6):38–44, November/December 2003.

[169] D. Pointcheval and J. Stern. Provably secure blind signature schemes. In M. Y. Rhee and K. Kim, editors, *Advances in Cryptology — Proceedings of ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 252–265. Springer-Verlag, 1996.

[170] G. J. Popek and C. S. Kline. Encryption and secure computer networks. *ACM Comput. Surv.*, 11(4):331–356, 1979.

[171] J. Postel and J. Reynolds. *RFC 959: File Transfer Protocol.* Internet Engineering Task Force, October 1985.

[172] S. Prabhakar, S. Pankanti, and A. K. Jain. Biometric recognition: Security and privacy concerns. *IEEE Security and Privacy*, 1(2):33–42, March-April 2003.

[173] M. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical Report LCS/TR-212, MIT Lab. for Computer Science, 1979.

[174] C. Radu. *Implementing Electronic Card Payment Systems.* Computer Security Series. Artech House, Norwood, 2002.

[175] A. J. Rae and L. P. Wildman. A taxonomy of attacks on secure devices. In J. Slay, editor, *Proceedings of the Fourth Australian Information Warfare and IT Security Conference*, pages 251–263, 2003.

[176] K. Rannenberg. Identity management in mobile cellular networks and related applications. *Information Security Technical Report*, 9(1):77–85, January–March 2004.

[177] J.-F. Raymond. Traffic analysis: Protocols, attacks, design issues, and open problems. In H. Federrath, editor, *Designing Privacy Enhancing Technologies, International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA,*

USA, July 25-26, 2000, Proceedings, volume 2009 of *Lecture Notes in Computer Science*, pages 10–29. Springer-Verlag, Berlin, 2001.

[178] M. Rennhard and B. Plattner. Introducing MorphMix: Peer-to-Peer based Anonymous Internet Usage with Collusion Detection. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002. ACM.

[179] E. Rescorla. *HTTP Over TLS*, 2000.

[180] E. Rescorla. *SSL and TLS*. Addison-Wesley, Reading, Massachusetts, 2001.

[181] V. Samar. Single sign-on using cookies for web applications. In *IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 158–164. IEEE Press, 1999.

[182] F. Satoh and T. Itoh. Single sign on architecture with dynamic tokens. In *Proceedings of the 2004 International Symposium on Applications and the Internet (SAINT'04)*, pages 197–200. IEEE Press, 2004.

[183] A. Serjantov. On the anonymity of anonymity systems. Technical Report UCAM-CL-TR-604, Computer Laboratory, University of Cambridge, U.K., October 2004.

[184] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In R. Dingledine and P. F. Syverson, editors, *Privacy Enhancing Technologies, Second International Workshop, PET 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers*, volume 2482 of *Lecture Notes in Computer Science*, pages 41–53. Springer-Verlag, Berlin, 2002.

[185] G. J. Simmons. Symmetric and asymmetric encryption. *ACM Comput. Surv.*, 11(4):305–330, 1979.

[186] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[187] M. Small. Business and technical motivation for identity management. *Information Security Technical Report*, 9(1):6–21, January–March 2004.

[188] N. Smart. *Cryptography, An Introduction*. McGraw-Hill, 2002.

[189] I. Spagui. *Secured Single Signon in a Client/Server Environment*. Vervante Corporate Publishing, 1994.

[190] W. Stallings. *Cryptography and network security (2nd ed.): principles and practice.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[191] S. Steinbrecher and S. Koepsell. Modelling unlinkability. In R. Dingledine, editor, *Privacy Enhancing Technologies, Third International Workshop, PET 2003, Dresden, Germany, March 26-28, 2003, Revised Papers*, volume 2760 of *Lecture Notes in Computer Science*, pages 32–47. Springer-Verlag, Berlin, 2003.

[192] J. G. Steiner, B. C. Neuman, and J. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 Usenix Conference*, pages 191–201. Usenix, February 1988.

[193] R. J. Sutton. *Secure Communications: Applications and Management.* John Wiley & Sons, 2002.

[194] P. F. Syverson and P. C. V. Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the IEEE Computer Security Foundations Workshop VII*, pages 14–29. IEEE Computer Society Press, 1994.

[195] N. T. Trask and M. V. Meyerstein. Smart cards in electronic commerce. *BT Technology Journal*, 17(3):57–66, July 1999.

[196] Trusted Computing Group. *TCG TPM Specification Version. 1.2 — Structures of the TPM*, 2003.

[197] Trusted Computing Group. *TCG TPM Specification Version. 1.2 — TPM Commands*, 2003.

[198] Trusted Computing Group. *TCG TPM Specification Version. 1.2 Design Principles*, 2003.

[199] U. Uludag and A. Jain. Attacks on biometric systems: a case study in fingerprints. In *Proceedings of SPIE-EI 2004*, pages 622–633, San Jose, CA, January 2004. SPIE.

[200] K. Vedder. GSM: Security, services, and the SIM. In B. Preneel and V. Rijmen, editors, *State of the Art in Applied Cryptography*, volume 1528 of *Lecture Notes in Computer Science*, pages 224–240. Springer-Verlag, Berlin, 1997.

[201] E. R. Verheul. Self-blindable credential certificates from the Weil pairing. In C. Boyd, editor, *ASIACRYPT '01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security*, volume 2248 of *Lecture Notes in Computer Science*, pages 533–551. Springer Verlag, Berlin, 2001.

[202] A. Volchkov. Revisiting single sign-on: A pragmatic approach in a new context. *IT Professional*, 3(1):39–45, January/February 2001.

[203] M. Walker and T. Wright. Security. In F. Hillebrand, editor, *GSM and UMTS: The creation of global mobile communication*, chapter 14, pages 385–406. John Wiley & Sons, 2002.

[204] J. Wayman, A. K. Jain, D. Maltoni, and D. Maio. *Biometric Systems: Technology, Design and Performance Evaluation*. Springer Verlag, 2005.

[205] M. J. Williamson. Thoughts on cheaper non-secret encryption. Report, CESG, August 1976.

[206] J. D. Woodward Jr., N. M. Orlans, and P. T. Higgins. *Biometrics: Identity Assurance In The Information Age*. McGraw Hill, January 2003.

[207] World Wide Web Consortium. *The Platform for Privacy Preferences 1.0 (P3P 1.0) Specification*, April 2002.

Part IV

Appendices

Appendix A

Impostor source code

This appendix provides the source code of Impostor, the SSO scheme described in chapter 7.

The Impostor implementation is divided into `Java` classes. Each of the subsections below contains the source code of one of these classes.

A.1 ChallengeResponseManager

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package impostor;

/**
  *   This interface defines the methods that a class must implement in order to
      provide the
  *   functionality of creating challenges for the Impostor daemon and verifying
      responses that
  *   come back from users.
  */
public interface ChallengeResponseManager
{
    /**
      *  The Impostor daemon calls this method in order to get the next challenge for
      *  carrying out user authentication.  Note that it is independent of any
          particular
```

```
     *  user since the challenge is issued BEFORE the user has a chance to
     *  identify him/herself.
     */
    Object getNewChallenge();

    /**
     *  The Impostor daemon calls this method in order to determine whether or not a
          given
     *  user identifier is valid (known). At the moment, the identifier
     *  is a String object that is retrieved from the user input of the Impostor
         login
     *  page (login.html).
     */
    boolean isValidIdentifier(Object identifier);

    /**
     *  The Impostor daemon calls this method in order to verify whether or not the
     *  response from the user identified by the given identifier matches the
     *  given challenge. The challenge object had been previously acquired
     *  using the getNewChallenge method. The identifier and response parameters
     *  are String objects, as entered by the user into the Impostor login page (
         login.html).
     */
    boolean verifyResponse(Object identifier, Object challenge, Object response);
}
```

A.2 ContentFilter

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package impostor;

/**
  *   This interface defines the methods that a class must implement in order to
      provide the
  *   functionality of looking for, and removing, sensitive material from HTTP
      headers and HTML pages
  *   sent from websites back to the user's browser.
  */
public interface ContentFilter
{
    /**
     *  The Impostor daemon calls this method in order to remove sensitive
         information
     *  from HTTP headers sent from websites to the user's browser.
     */
    String filterHTTPHeaders(String headers);
```

```
    /**
      * The Impostor daemon calls this method in order to remove sensitive
          information
      * from HTML pages sent from websites to the user's browser. The method is
          called on
          * a line-by-line basis.
      */
    String filterWebPageLine(String currentLine);
}
```

A.3 EmptyManager

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package impostor;

/** This class provides simple or empty implementations of the interfaces of the <
    code>impostor</code> package.
  * It is used internally by an Impostor daemon that is instantiated without all
      arguments.
  */
class EmptyManager implements ContentFilter, ChallengeResponseManager, UserManager,
    RequestRecognizer

{
    // Simple or null implementations of Impostor interfaces

    public String filterHTTPHeaders(String string) {return string;}
    public String filterWebPageLine(String string) {return string;}
    public Object getNewChallenge(){return null;}
    public boolean isValidIdentifier(Object identifier){return false;}
    public boolean verifyResponse(Object identifier, Object challenge, Object
        response){return false;}
    public String getUsernameForIdentifier(Object identifier, RequestRecognizer rr)
        throws Exception{return null;}
    public String getPasswordForIdentifier(Object identifier, RequestRecognizer rr)
        throws Exception{return null;}
    public RequestRecognizer getRequestRecognizerInstance(){return null;}
    public void init(String host, int port, String request){};
    public boolean isRecognized(){return false;}
    public String getServiceName(){return null;}
    public String fillInUsernameAndPassword(String username, String password){return
        null;}
    public String getLogEntry(){return null;}

}
```

A.4 Impostor

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package impostor;


import java.net.*;
import java.io.*;
import java.security.*;
import java.security.cert.Certificate;
import javax.net.ssl.*;
import java.util.Vector;

/**
  *   This class implements a simple HTTP proxy daemon with Single Sign-On
      functionality into websites. If it
  *   is instantiated properly, it recognizes HTTP requests that constitute a user's
      login request into
  *   a website. The Impostor daemon then invokes a Challenge-Response authentication
       mechanism in order to
  *   authenticate the user. If successful, it automatically fills in the username
      and password of the
  *   authenticated user for the website that is being visited. In this way the user
      will not have to remember
  *   multiple usernames and passwords for different websites, and will also be able
      to log into the
  *   websites without having to type his/her password into the access device he/she
      is using (this could be,
  *   for example, an untrusted device in an Internet cafe) . The Impostor daemon
      also "intercepts" SSL/TLS connections by setting
  *   up two separate SSL connections between the user's browser and the visited
      website. This way the daemon
  *   is able to extract the HTTP requests that are sent over the SSL/TLS connection.
       <br> <br>
  *
  *   The daemon needs to have an asymmetric keypair and a certificate for its public
       key in order to be able to set up
  *   SSL connections. This data is expected to be found in a keystore file named <
      code>prvkey</code> and the keystore
  *   password (as well as the alias password) is expected to be "<code>secret</code
      >". An easy way to create this
  *   keystore file is using the command <code>keytool -genkey -keyalg RSA -keysize
      1024 -keystore prvkey</code> and
  *   typing <code>secret</code> whenever asked for a password. <br> <br>
  *
  *   The daemon also needs access to two html pages, which are expected to be found
      in files named <code>login.html</code> and
  *   <code>error.html</code>. These pages need to be constructed according to some
```

```
     simple guidelines and will serve as
  *  the Impostor login and error pages respectively. <br><br>
  *
  *  This class extends the {@link java.lang.Thread} class, which means that the
     start method should be called
  *  in order for the web proxy daemon to actually start serving incoming
     connections. If it is desired to stop
  *  the daemon, the <code>shutdown</code> method should be used.
  *
  *  @author      Andreas Pashalidis
  */

public final class Impostor extends Thread
{
    protected static final String NAME="Impostor v.0.9";
    private static final String CONSOLENOTICE="\n           Impostor - written by
        Andreas Pashalidis\n           ------------------------------------\nThis
        software is provided \"as is\" without warranty of any kind.\nNo
        responsibility will be accepted for any negative effects the\nsoftware may
        cause. You use it on your own risk.\n("+NAME+")\n";
    private static final String KEYSTOREFILENAME="prvkey";
    private static final String KEYSTOREPASSWORD="secret";
    private static final java.text.DateFormat df = java.text.DateFormat.
        getDateTimeInstance(3,2);

    private PrintWriter log;
    private SSLContext sslc;
    private ChallengeResponseManager crm;
    private UserManager um;
    private ContentFilter cf;
    private ServerSocket ss;
    private boolean isActive;

    protected Vector sensitiveStrings;
    protected Vector replacementStrings;

    /** Creates an Impostor web proxy that will run on port 8080 and has no extra
        functionality. Log messages will be sent to standard output.*/
    public Impostor() throws Exception
    {
        this(8080, new PrintWriter(System.out, true), new EmptyManager(), new
            EmptyManager(), new EmptyManager());
    }

    /** Creates an Impostor web proxy that will run on the specified port but no
        extra functionality. Log messages will be sent to standard output.*/
    public Impostor(int port) throws Exception
    {
        this(port, new PrintWriter(System.out, true), new EmptyManager(), new
            EmptyManager(), new EmptyManager());
    }

    /** Creates an Impostor web proxy that will run on the specified port but no
        extra functionality. Log messages will be sent to the specified {@link
        PrintWriter}.*/
    public Impostor(int port, PrintWriter log) throws Exception
    {
        this(port, log, new EmptyManager(), new EmptyManager(), new EmptyManager());
    }

    /**  Creates an Impostor web proxy that will run on the specified port, send log
        messages to the specified {@link PrintWriter} and will use
      *  the specified {@link ContentFilter}.
      */
    public Impostor(int port, PrintWriter log, ContentFilter cf) throws Exception
    {
```

```
    this(port, log, new EmptyManager(), new EmptyManager(), cf);
}

/**
  *   This is the full constructor that provides the maximum flexibility and
      functionality. It creates an Impostor web proxy
  *   that will run on the specified port and sent log messages to the specified
      {@link PrintWriter}. It also allows
  *   the caller to specify implementations of a {@link UserManager}, a {@link
      ChallengeResponseManager} and a
  *   {@link ContentFilter}.
  *
  *   @param port       the port the Impostor proxy shall run on
  *   @param log        the {@link PrintWriter} log messages shall be sent to
  *   @param um         the {@link UserManager} implementation the Impostor shall
       use
  *   @param crm        the {@link ChallengeResponseManager} implementation the
      Impostor shall use
  *   @param cf         the {@link ContentFilter} implementation the Impostor
      shall use
  */
public Impostor(int port, PrintWriter log, UserManager um,
    ChallengeResponseManager crm, ContentFilter cf) throws Exception
{
    System.out.println(CONSOLENOTICE);
    if (port<1 || port>65535) throw new IllegalArgumentException("Not a valid
        port number: "+port);
    if (log==null) throw new NullPointerException("Logger is null!");
    if (um==null) throw new NullPointerException("UserMaager is null!");
    if (crm==null) throw new NullPointerException("ChallengeResponseManager is
        null!");
    if (cf==null) throw new NullPointerException("ContentFilter is null!");
    this.log=log;
    this.um=um;
    this.crm=crm;
    this.cf=cf;
    log("Initializing SSL Environment...");
    KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
    ks.load(new FileInputStream(KEYSTOREFILENAME),null);

    KeyManagerFactory kmf=KeyManagerFactory.getInstance(KeyManagerFactory.
        getDefaultAlgorithm());
    kmf.init(ks, KEYSTOREPASSWORD.toCharArray());
    KeyManager[] kms = kmf.getKeyManagers();
    sslc=SSLContext.getInstance("SSL");
    sslc.init(kmf.getKeyManagers(), null, null);
    ss=new ServerSocket();
    setDaemon(true);
    log("Attempting to start "+NAME+" on port "+port+"...");
    ss=new ServerSocket(port);
    isActive=false;
    log(NAME+" is ready...");
}

/**
  *   This method has to be called in order for the Impostor web proxy to start.
  */

public final void run()
{
    isActive=true;
    log(NAME+" is running.");
    while (isActive)
    {
        try
        {
```

```
                new Servant(this, ss.accept());
            } catch (IOException e)
            {
//                e.printStackTrace();
                log("Fatal I/O Exception: "+e.getMessage());
            }
        }
    }

    /**
      * As the stop method in {@link Thread} is deprecated, this method should be
        called in order
      * to properly stop a running Impostor web proxy.
      */
    public final void shutdown() throws Exception
    {
        if (!isActive) return;
        log("Attempting to stop "+NAME+" ...");
        ss.close();
        isActive=false;
        join();
        log(NAME+" stopped.");
    }

    protected final void log(String s){log.println("["+df.format(new java.util.Date()
        )+"] "+s);log.flush();}
    protected final SSLContext getSSLContext(){return sslc;}
    protected final SSLSocketFactory getSSLSocketFactory(){return sslc.
        getSocketFactory();}
    protected final SSLServerSocketFactory getSSLServerSocketFactory(){return sslc.
        getServerSocketFactory();}
    protected final ChallengeResponseManager getChallengeResponseManager(){return crm
        ;}
    protected final UserManager getUserManager(){return um;}
    protected final ContentFilter getContentFilter(){return cf;}


    protected final static String readInputStream(InputStream is) throws IOException
    {
        byte[] reply = new byte[65536];
        int size=is.read(reply);
        if (size>0) return new String(reply, 0, size); else return null;
    }
}
```

A.5 LoginHandler

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
```

```
*/

package impostor;


import java.util.Vector;
import java.io.*;
import java.net.Socket;

final class LoginHandler
{
    private static final int tolerance=60; // how many seconds tolerance?
    private static final String LOGINPAGEFILENAME="login.html";

    private static java.security.SecureRandom sr = new java.security.SecureRandom();

    private Long handle;
    private Object challenge;
    private Socket server;
    private RequestRecognizer rr;

    protected LoginHandler(Socket server, RequestRecognizer rr, Object challenge)
        throws java.net.SocketException, FileNotFoundException, IOException
    {
        server.setKeepAlive(true);
        this.server=server;
        this.rr=rr;
        this.handle=new Long(System.currentTimeMillis());
        this.challenge = challenge;
    }

    protected Long getHandle(){return handle;}
    protected Object getChallenge(){return challenge;}

    protected boolean isValid()
    {
        long lastInstant=handle.longValue()+tolerance*1000;
        return (System.currentTimeMillis()<lastInstant);
    }

    protected Socket getServerSocket(){return server;}

    protected RequestRecognizer getRequestRecognizer()
    {
        return rr;
    }

    protected String makeLoginPage(String currentSite) throws IOException
    {
        String page="HTTP/1.0 200 OK\r\nServer: "+Impostor.NAME+"\r\nContent-Type:
            text/html\r\n\r\n";

        BufferedReader br = new BufferedReader(new FileReader(LOGINPAGEFILENAME));
        while (br.ready()) page+=br.readLine();
        br.close();
        page=page.replaceAll("%Challenge%", challenge.toString());
        page=page.replaceAll("%handleString%", handle.toString());
        page=page.replaceAll("%serviceName%", rr.getServiceName());
        page=page.replaceAll("%currentSite%", currentSite);
        return page;
    }

}
```

A.6 RequestHandler

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package impostor;

import java.net.*;
import java.io.*;
import java.util.StringTokenizer;

/** Internal Impostor class */
final class RequestHandler
{
    private String headers, postString;
    private String scheme, host;
    private int port;
    private boolean isConnect=false, isGet=false, isPost=false, isLocal=false;

    protected RequestHandler(InputStream is) throws IOException
    {
        headers=readHTTPHeaders(is);
        // we don't support persistent connections; so we downgrade to HTTP/1.0
        headers=java.util.regex.Pattern.compile("accept-encoding:␣[^\r]*", 2).matcher
            (headers).replaceFirst("Accept-Encoding:␣Identity");
        headers=java.util.regex.Pattern.compile("HTTP/1.1").matcher(headers).
            replaceFirst("HTTP/1.0");
        headers=java.util.regex.Pattern.compile("connection:␣keep-alive", 2).matcher(
            headers).replaceFirst("Connection:␣Close");
        StringTokenizer st = new StringTokenizer(headers);
        String action=st.nextToken();
        if (action.equals("CONNECT"))
        {
            isConnect=true;
            examineURL(st.nextToken());

        } else
        if (action.equals("POST"))
        {
            isPost=true;
            examineURL(st.nextToken());
            headers=headers.replaceFirst(scheme+"://"+host+":"+port, "");
            headers=headers.replaceFirst(scheme+"://"+host, "");
            postString=Impostor.readInputStream(is);
            // check whether the user just completed the Impostor Login Form
            java.util.regex.Matcher m = java.util.regex.Pattern.compile("impostor=
                impostor.sec").matcher(postString);
            isLocal=m.find();
        } else
        {
```

```
            isGet=true;
            examineURL(st.nextToken());

            headers=headers.replaceFirst(scheme+"://"+host+":"+port, "");
            headers=headers.replaceFirst(scheme+"://"+host, "");
        }
    }


    private void examineURL(String url)
    {
        port=-1;

        // look for the "://" substring (which means that we have the scheme)
        java.util.regex.Matcher m;
        m=java.util.regex.Pattern.compile("://").matcher(url);
        boolean hasScheme=m.find();

        if (hasScheme)
        {
            scheme=url.substring(0, m.start());
            m=java.util.regex.Pattern.compile("://[^:/]*").matcher(url);
            if (!m.find()) throw new RuntimeException("Could not parse the hostname
                out of this url ("+url+")");
            host=url.substring(m.start()+3, m.end());
            if (url.substring(m.end(), m.end()+1).equals(":"))
            {
                java.util.StringTokenizer st = new java.util.StringTokenizer(url.
                    substring(m.end()+1), "/", false);
                port=Integer.parseInt(st.nextToken());
            }
            if (port<0)
            {
                if (scheme.equals("http")) port=80;
                if (scheme.equals("https")) port=447;
            }
        } else
        {
            // we dont have the scheme, so scheme = null
            // we are expecting host and port only if this is a connect request! (
                whithout this check we get NullPointerExceptions)
            if (isConnect)
            {
                StringTokenizer st=new StringTokenizer(url,":",false);
                host=st.nextToken();
                port=Integer.parseInt(st.nextToken());
            }
            // if this is NOT a connect request it contains probably not the host
            // (is it probably a GET request for a relative URL)
        }
    }


    protected String getRequest(){if (postString==null) return headers; else return
        headers+postString;}
    protected String getHeaders(){return headers;}
    protected String getPostString(){return postString;}
    protected String getScheme(){return scheme;}
    protected String getHost(){return host;}
    protected int getPort(){return port;}
    protected boolean isLocal(){return isLocal;}
    protected boolean isGet(){return isGet;}
    protected boolean isConnect(){return isConnect;}
    protected boolean isPost(){return isPost;}

    private static void say(String s) {System.out.println(s);}
```

```
    private static int getPort(URL url){int temp=url.getPort();if (temp<10) return
        url.getDefaultPort(); else return temp;}

    protected static String readHTTPHeaders(InputStream is) throws IOException
    {
        byte[] reply = new byte[4096];
        int replyLen = 0;
        int newlinesSeen = 0;
        while (newlinesSeen < 2)
        {
            int i = is.read();
            if (i < 0) throw new IOException("Unexpected EOF after ("+new String(
                reply, 0, replyLen)+")");
            if (i == '\n')
            {
                newlinesSeen++;
            } else
            if (i != '\r')
            {
                newlinesSeen = 0;
            }
            reply[replyLen++] = (byte) i;
        }
        return new String(reply, 0, replyLen);
    }
}
```

A.7 RequestRecognizer

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package impostor;

/**
  *   This interface defines the methods that a class must implement in order to
      provide the
  *   functionality of recognizing HTML requests sent from the user's browser to
      websites.
  *   It is the basic functionality that enables the Impostor daemon to provide
      Single Sign-On.
  *   For every incoming HTML request (even those over SSL/TLS connections) the
      Impostor daemon first acquires a
  *   RequestRecognizer instance
  *   using a {@link UserManager}'s <code>getRequestRecognizerInstance</code> method.
       It then calls the RequestRecognizer's
  *  <code>init</code> method in order to initialize the RequestRecognizer. If this
```

```
     request is recognized as
  *  a user's login attempt to the website, the <code>isRecognized</code> method
     should return true. If this
  *  is the case, the Impostor daemon calls the <code>getServiceName</code> and <
     code>getLogEntry</code> methods in order to
  *  generate the Impostor login page and a log entry. Assuming successful user
     authentication
  *  (according to the challenge/response mechanism implemented by the {@link
     ChallengeResponseManager}),
  *  the Impostor daemon calls the <code>fillInUsernameAndPassword</code> method
     specifying a specific username
  *  and password. The method should return a valid HTTP request that results in the
      user being
  *  logged into the website under the specified username.
  */
public interface RequestRecognizer
{
  /**
    *  The Impostor daemon calls this method in order to initialize the
       RequestRecognizer.
    *  It is called for every HTML request coming from a user. The parameters
    *  passed to the method is the host name of the website, the port number
    *  of the TCP socket (this is typically 80 for HTTP or 447 for HTTPS) and the
       HTTP
    *  request itself. This includes HTML request headers and, only in the case
    *  of a POST request, the POST string that follows immetiately after the
    *  headers. An implementing class should analyze the request and determine
    *  whether this is a login request for a website or not.
    */
    void init(String host, int port, String request);

  /**
    *  The Impostor daemon calls this method in order to determine whether this
       RequestRecognizer
    *  recognized the HTTP request with which it was initialized as a login attempt
       into a
    *  website. The method should return false if the <code>init</code> method of
       this object has not been
    *  called yet, or if the request was not recognized as a login attempt. If this
       method returns
    *  true, the Impostor daemon expects the remaining methods to return non-null
       values.
    */
    boolean isRecognized();

  /**
    *  The Impostor daemon calls this method only if this RequestRecognizer's <code>
       isRecognized</code> method returns
    *  true. This method should return the name of the service or website this
       recognized request
    *  is a login attempt for. The daemon uses this name in order to generate the
       Impostor login page.
    */
    String getServiceName();

  /**
    *  A RequestRecognizer must also implement the functionality to fill a given
       username
    *  and password into the HTTP request with which is was initialized, such that
       the resulting HTTP request effectively
    *  logs the specified username into the site, using the specified password. The
       Impostor daemon
    *  calls this method only if this RequestRecognizer's <code>isRecognized</code>
       method returns
    *  true. The daemon will call this method only after a valid Impostor user has
    *  successfully authenticated him/herself. (as determined by the imlementation of
```

```
         a
     *  {@link ChallengeResponseManager}). The username and password the Impostor
         daemon passes as parameters
     *  to this method are determined by the implementation of a {@link UserManager}.
     */
     String fillInUsernameAndPassword(String username, String password);

   /**
     *  The Impostor daemon calls this method in order to store an entry in a log for
         this
     *  RequestRecognizer. An implementation should return the details of an HTTP
         request
     *  if, of course, it has been initialized.
     */
     String getLogEntry();
}
```

A.8 Servant

```
/*
    This class is the implements the Servant thread for each incoming HTTP request;
        it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
package impostor;

import java.net.*;
import java.io.*;
import javax.net.ssl.*;
import java.util.Vector;

final class Servant extends Thread
{
    private static final String ERRORPAGEFILENAME="error.html";
    private static Vector loginHandlers=new Vector();
    private static long total=0;
    private long id;
    private Socket client;
    private Impostor imp;

    protected Servant(Impostor imp, Socket client)
    {
        this.client=client;
        this.imp=imp;
        total++;
        id=total;
        start();
    }
```

```
    public void run()
    {
        String clientString=client.getInetAddress().getHostName()+":"+client.getPort
            ();
        try
        {
            RequestHandler rh=new RequestHandler(client.getInputStream());
//              imp.log("Serving request from "+clientString+" ("+id+") "+rh.getRequest
    ());

            if (rh.isLocal())  // someone just completed a Impostor Logon Form
            {
                String request=rh.getRequest();
                java.util.regex.Matcher m;
                m = java.util.regex.Pattern.compile("identifier=[^&]*").matcher(
                    request);
                if (!m.find()) throw new RuntimeException("Could not find identifier
                    during Impostor Logon!");
                String identifier=m.group().substring(11);
                m = java.util.regex.Pattern.compile("challenge=[^&]*").matcher(
                    request);
                if (!m.find()) throw new RuntimeException("Could not find challenge
                    during Impostor Logon!");
                String challenge=m.group().substring(10);
                m = java.util.regex.Pattern.compile("response=[^&]*").matcher(request
                    );
                if (!m.find()) throw new RuntimeException("Could not find response
                    during Impostor Logon!");
                String response=m.group().substring(9);
                m = java.util.regex.Pattern.compile("handleString=[^&]*").matcher(
                    request);
                if (!m.find()) throw new RuntimeException("Could not find response
                    during Impostor Logon!");
                Long handle = new Long(m.group().substring(13));
                log("Impostor Login attempt with handle="+handle+", identifier="+
                    identifier+", challenge="+challenge+" and response="+response);
                LoginHandler lh=findLoginHandler(handle);
                String errorPage=null;
                if (lh==null)
                {
                    errorPage=makeErrorPage("The timestamp "+handle+" is not valid (
                        any more).");
                } else
                if (!imp.getChallengeResponseManager().isValidIdentifier(identifier))
                {
                    if (identifier.equals("")) errorPage=makeErrorPage("You did not
                        specify an identifier."); else
                    errorPage=makeErrorPage(identifier+" does not appear to be valid
                        on this system.");
                } else
                if (!imp.getChallengeResponseManager().verifyResponse(identifier, lh.
                    getChallenge(), response))
                {
                    errorPage=makeErrorPage("Access Denied.");
                }
                if (errorPage==null)
                {
                    RequestRecognizer rr = lh.getRequestRecognizer();
                    String username=imp.getUserManager().getUsernameForIdentifier(
                        identifier, rr);
                    String password=imp.getUserManager().getPasswordForIdentifier(
                        identifier, rr);
                    request=rr.fillInUsernameAndPassword(username, password);
                    log("logging "+identifier+" into "+rr.getServiceName()+" with
                        username="+username+" and password="+password);
```

```
//                  log("the request for this is:\n"+request);
                    Socket server=lh.getServerSocket();
                    OutputStream os = server.getOutputStream();
                    os.write(request.getBytes());
                    os.flush();
                    serverToClient(imp, server, client);
                } else
                {
                    OutputStream os=client.getOutputStream();
                    os.write(errorPage.getBytes());
                    os.flush();
                    client.close();
                }
            } else
            if (rh.isConnect()) // An SSL connection is needed, so let's set it up!
            {
                // first get a proper SSL socket factory.
                SSLSocketFactory sf=imp.getSSLSocketFactory();

                // now take an SSL socket and connect it to the remote web server
                SSLSocket server=(SSLSocket)sf.createSocket(rh.getHost(),rh.getPort()
                    );
                server.setUseClientMode(true);
                server.startHandshake();

                // now say to the client that we are ready to upgrade to SSL
                OutputStream os;
                os=client.getOutputStream();
                String proxyResponse="HTTP/1.1 200 Connection Established\r\n\r\n";
                os.write(proxyResponse.getBytes());
                os.flush();

                // now do actually the upgrade to SSL
                String clientHost=client.getInetAddress().getCanonicalHostName();
                int    clientPort=client.getPort();

                SSLSocket SSLClient=(SSLSocket)sf.createSocket(client, clientHost,
                    clientPort, true);
                SSLClient.setUseClientMode(false); // we are the server of this SSL
                    socket
                SSLClient.startHandshake();

                // now get the new HTTP request from the client
                RequestHandler SSLrh=new RequestHandler(SSLClient.getInputStream());
                String request=SSLrh.getRequest();
//                imp.log("Serving SSL request from "+clientString+" ("+id+") "+
    request);
                RequestRecognizer rr = imp.getUserManager().
                    getRequestRecognizerInstance();
                rr.init(rh.getHost(), rh.getPort(), request);


                if (rr.isRecognized())
                {
                    log(clientString+": "+rr.getLogEntry());
                    // now we have to ask for impostor login before filling in the
                        password
                    LoginHandler lh = new LoginHandler(server, rr, imp.
                        getChallengeResponseManager().getNewChallenge());
                    showLoginPage(SSLClient, lh, "http://"+rh.getHost());
                    return;
                }

                // forward the request to the server
                os=server.getOutputStream();
                os.write(request.getBytes());
```

```
                os.flush();

                serverToClient(imp, server, SSLClient);
            } else
            // we have an HTTP GET
            {
                String request = rh.getRequest();
                Socket server=new Socket(rh.getHost(), rh.getPort());
                RequestRecognizer rr = imp.getUserManager().
                    getRequestRecognizerInstance();
                rr.init(rh.getHost(), rh.getPort(), request);
                if (rr.isRecognized())
                {
                    log(clientString+": "+rr.getLogEntry());
                    // now we have to ask for impostor login before filling in the
                        password
                    LoginHandler lh = new LoginHandler(server, rr, imp.
                        getChallengeResponseManager().getNewChallenge());
                    showLoginPage(client, lh, "http://"+rh.getHost());
                    return;
                }
                OutputStream os=server.getOutputStream();
                os.write(request.getBytes());
                os.flush();
                serverToClient(imp, server, client);
            }


        } catch (Exception e)
        {
//            e.printStackTrace();
            log("Error: "+clientString+" "+e.getMessage()+" ("+id+").");
            return;
        }
    }


    private static void showLoginPage(Socket client, LoginHandler lh, String
        currentSite) throws IOException
    {
        loginHandlers.addElement(lh);
        OutputStream os = client.getOutputStream();
        os.write(lh.makeLoginPage(currentSite).getBytes());
        os.flush();
        client.close();
    }

    private static LoginHandler findLoginHandler(Long handle)
    {
        LoginHandler lh=null;
        java.util.Enumeration e=loginHandlers.elements();
        while (e.hasMoreElements())
        {
            LoginHandler test=(LoginHandler)e.nextElement();
            if (!test.isValid())
            {
                loginHandlers.removeElement(test);
            } else
            if (test.getHandle().equals(handle))
            {
                lh=test;
            }
        }
        loginHandlers.remove(lh);
        return lh;
    }
```

```
    private static void serverToClient(Impostor imp, Socket server, Socket client)
        throws IOException, InterruptedException
    {
        InputStream is  = server.getInputStream();
        OutputStream os = client.getOutputStream();
        String headers=RequestHandler.readHTTPHeaders(is);
        java.util.regex.Matcher m;
        m = java.util.regex.Pattern.compile("connection: keep-alive", 2).matcher(
            headers);
        headers=m.replaceFirst("Connection: Close");
        ContentFilter cf=imp.getContentFilter();
        headers=cf.filterHTTPHeaders(headers);
//        System.out.println("Filtered Response Headers:\n"+headers);
        os.write(headers.getBytes());
        os.flush();
        m = java.util.regex.Pattern.compile("Content-Type: text/html", 2).matcher(
            headers);
        if (m.find())
        {

            BufferedReader br = new BufferedReader(new InputStreamReader(is));
            String thisLine=br.readLine();
            while (thisLine!=null)
            {
                thisLine=cf.filterWebPageLine(thisLine)+"\r\n";
                os.write(thisLine.getBytes());
                os.flush();
                thisLine=br.readLine();
            }
            os.close();
            br.close();
            return;
        }
        boolean isActive=true;
        byte[] buffer=new byte[32768];
        int b;
        while (isActive)
        {
            b=is.read(buffer);
            if (b<0)
            {
                isActive=false;
                os.close();
            } else
            {
                os.write(buffer,0,b);
                os.flush();
            }
        }
    }


    private void log(String s){imp.log(id+" - "+s);}

    protected static String makeErrorPage(String error) throws IOException
    {
        String page="HTTP/1.1 200 OK\r\nServer: "+Impostor.NAME+"\r\nContent-Type:
            text/html\r\n\r\n";

        BufferedReader br = new BufferedReader(new FileReader(ERRORPAGEFILENAME));
        while (br.ready()) page+=br.readLine();
        br.close();
        page=page.replaceAll("%error%", error);
```

```
        return page;
    }

}
```

A.9 UserManager

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package impostor;


/**
  *   This interface defines the methods that a class must implement in order to
      provide the
  *   functionality of mapping a given Impostor user to its username/password pairs
      he/she
  *   maintains at the websites for which Single Sign-On is to be achieved. A
      UserManager
  *   implementation works very close with a {@link RequestRecognizer} implementation
      : the RequestRecognizer
  *   recognizes HTTP requests as login attemps into a specific set of websites,
      while
  *   the UserManager knows all Impostor users' usernames and passwords for exactly
      this
  *   set of websites. A UserManager implementation must also offer a method to
      create
  *   instances of the {@link RequestRecognizer} implementation this UserManager
      works with.
  *
  */

public interface UserManager
{

    /**
      *  The Impostor daemon calls this method in order to determine the username
          that the Impostor user
      *  identified by the given identifier maintains at the website for which the
          given {@link RequestRecognizer}
      *  recognized a login request. The identifier is a String object, obtained from
           the user
      *  input from the Impostor login page, and the RequestRecognizer is the
          RequestRecognizer
      *  instance that recognized the HTTP request as a login request into a website
          .<br>
```

```
     *
     *   Note that before calling this method the Impostor daemon checks the
         validity of the user identifier using the <code>isValid</code> method
     *   of the {@link ChallengeResponseManager} implementation with which the
         daemon was instantiated
     *   Thus, a UserManager and a ChallengeResponseManager have common user
         identifiers: the
     *   identifiers of Impostor users.
     */
    String getUsernameForIdentifier(Object identifier, RequestRecognizer rr) throws
        Exception;

    /**
     * The Impostor daemon calls this method in order to determine the password
         that the Impostor user
     * identified by the given identifier maintains at the website for which the
         given {@link RequestRecognizer}
     * recognized a login request. The identifier is a String object, obtained from
          the user
     * input from the Impostor login page, and the RequestRecognizer is a
         RequestRecognizer
     * implementation previously obtained using the getRequestRecognizerInstance
         method.
     */
    String getPasswordForIdentifier(Object identifier, RequestRecognizer rr) throws
        Exception;

    /**
     * A UserManager implementation has to work very close with a {@link
         RequestRecognizer} implementation:
     * the UserManager implementation knows the usernames and passwords of Impostor
          users at a specific
     * set of websites, while the RequestRecognizer implementation recognizes HTTP
         login requests
     * for exactly this set of websites. This method should return a new
         RequestRecognizer
     * instance of the RequestRecognizer this UserManager implementation works with
         . The Impostor daemon
     * calls this method for every incoming HTTP request, as it needs a fresh
         RequestRecognizer.<br>
     *
     *   Note that before calling this method the Impostor daemon checks the
         validity of the user identifier using the <code>isValid</code> method
     *   of the {@link ChallengeResponseManager} implementation with which the
         Impostor was instantiated.
     *   Thus, a UserManager and a ChallengeResponseManager have common user
         identifiers: the
     *   identifiers of Impostor users.
     */
    RequestRecognizer getRequestRecognizerInstance();
}
```

Appendix B

Source code of peer-to-peer system

This appendix provides the source code of the peer-to-peer synchronisation scheme described in chapter 10.

The implementation of the scheme is divided into `Java` classes. Each of the sections below contains the source code of a number of classes, grouped according to functionality they provide.

B.1 Protocol Messages

This section provides the source code of the messages that are used by the protocols of the scheme.

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package protocols;


import java.net.*;
import java.io.*;
```

```
/**
  *   This class defines a structure for contacting Internet hosts.
  *   It is a holder for an java.net.InetAddress and a port number.
  *
  *   @author Andreas Pashalidis
  */

public final class IpAndPort implements Serializable
{
    private java.net.InetAddress ipaddr;
    private int port;
    public IpAndPort(java.net.InetAddress ad, int port)
    {
        this.ipaddr=ad;
        this.port=port;
    }

    public java.net.InetAddress getAddr()
    {
        return ipaddr;
    }
    public int getPort()
    {
        return port;
    }

    public boolean equals(IpAndPort adr)
    {
        if (adr.getPort()!=port) return false;
        if (!(adr.getAddr().getHostAddress().equals(ipaddr.getHostAddress()))) return
            false;
        return true;
    }

    public String toString()
    {
        return ipaddr.getHostAddress()+":"+port;
    }
}

/**
  *   This class is the superclass of all protocol Messages.
  */

package protocols;

public class Message extends Object implements java.io.Serializable
{
    public int version;
}

/**
  *   This class represent a generic protocol message that indicates
  *   success or failure, from the sender's point of view.
  */

package protocols;

public class MessageBinary extends Message
{
    public boolean success;
    public String  errorMessage;

    public MessageBinary(boolean success, String errorMessage)
    {
        version=1;
```

```
        this.success=success;
        this.errorMessage=errorMessage;
    }
}

/**
  *   This class represent a generic protocol message that indicates
  *   success or failure, from the sender's point of view.
  */

package protocols.discover;

public class DiscoveryRequest extends protocols.Message
{
    public int low, high;


    public DiscoveryRequest(int low, int high)
    {
        version=1;
        this.low=low;
        this.high=high;
    }
}

/**
  *   This class represent a generic protocol message that indicates
  *   success or failure, from the sender's point of view.
  */

package protocols.discover;

public class DiscoveryReply extends protocols.Message
{
    public java.util.Vector GMRecords;
    public DiscoveryReply(java.util.Vector gmr)
    {
        version=1;
        this.GMRecords=gmr;
    }
}

/**
  *   This class represent a generic protocol message that indicates
  *   success or failure, from the sender's point of view.
  */

package protocols.echo;

public class MessageEcho extends protocols.Message
{
    public boolean echo;
    public long time;

    public MessageEcho(boolean echo, long time)
    {
        version=1;
        this.echo=echo;
        this.time=time;
    }
}

/**
  *   This class represent a generic protocol message that indicates
  *   success or failure, from the sender's point of view.
```

```
  */

package protocols.join;

public class JoinRequest extends protocols.Message
{
    public int min, max, port;
    public JoinRequest(int min, int max, int port)
    {
        version=1;
        this.min=min;
        this.max=max;
        this.port=port;
    }
}

/**
  *   This class represent a generic protocol message that indicates
  *   success or failure, from the sender's point of view.
  */

package protocols.join;

public class JoinReply extends protocols.Message
{
    public boolean success;
    public java.util.Vector userRecords;
    public String errorMessage;
    public int groupSize;
    public protocols.IpAndPort ipap;

    public JoinReply(boolean success, String errorMessage, java.util.Vector
        userRecords, int groupSize, protocols.IpAndPort ipap)
    {
        version=1;
        this.success=success;
        this.errorMessage=errorMessage;
        this.userRecords=userRecords;
        this.groupSize=groupSize;
        this.ipap=ipap;
    }
}

/**
  *   This class represent protocol message 1 of the register protocol.
  *   It is sent from a Group Manager to a Discovery Server. Its purpose
  *   is for the Discovery Server to list the Group Manager in his directory.
  */

package protocols.register;

public class RegisterRequest extends protocols.Message
{
    public int groupSize;
    public int port;

    public RegisterRequest(int groupSize, int port)
    {
        version=1;
        this.groupSize=groupSize;
        this.port=port;
    }
}
```

B.2 Plain Member

This section provides the client source code, i.e. the implementation of the Plain group Member (PM) role.

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package client;


import java.net.*;
import java.io.*;
import protocols.IpAndPort;
import javax.swing.event.EventListenerList;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;

/**
  *   This class implements the functionality of the Group Manager daemon.
  *
  *   @author Andreas Pashalidis
  */

public final class Client extends Thread implements pinger.PingingDaemon
{
    protected static final String NAME="Client Daemon v.0.2";
    private static final String CONSOLENOTICE="\n          ClientDaemon - written by
        Andreas Pashalidis\n          ----------\nThis
        software is provided \"as is\" without warranty of any kind.\nNo
        responsibility will be accepted for any negative effects the\nsoftware may
        cause. You use it on your own risk.\n("+NAME+")\n";
    private static final java.text.DateFormat df = java.text.DateFormat.
        getDateTimeInstance(3,2);
    private EventListenerList listeners;
    private volatile java.util.Vector ptList;
    private boolean pingingEnabled;
    private GMConnectionManager gmcm;
    private IpAndPort myIpap;


    private PrintWriter log;
    private ServerSocket ss; // client server
    private boolean isActive;


    /** Creates a Client that will log messages to the standard output.*/
    public Client(int port) throws Exception
```

```
{
    this(port, new PrintWriter(System.out, true));
}


/**
  *   This is the full constructor that provides the maximum flexibility and
      functionality. It creates a Client
  *   that will log messages to the specified {@link PrintWriter}.
  *
  *   @param init       the initialization object for this Group Manager instance
  *   @param log        the {@link PrintWriter} log messages shall be sent to
  */
public Client(int port, PrintWriter log) throws Exception
{
    if (port<1 || port>65535) throw new IllegalArgumentException("Not a valid
        port number:"+port);
    listeners=new EventListenerList();
    ptList=new java.util.Vector();
    System.out.println(CONSOLENOTICE);
    if (log==null) throw new NullPointerException("Logger is null!");
    this.log=log;
    setDaemon(true);
    pingingEnabled=false;
    myIpap=null;
    log("Attempting to start "+NAME+" on port "+port+"...");
    ss=new ServerSocket(port);
    log(NAME+" is ready.");
}

public void register(int min, int max,IpAndPort GMAddr) throws Exception
{
    log("Registering at GroupManager...");
    Socket gmSocket=new Socket();
    gmSocket.setTcpNoDelay(true);
    gmSocket.setSoTimeout(10000);
    gmSocket.connect(new InetSocketAddress(GMAddr.getAddr(),GMAddr.getPort())
        ,10000);
    ObjectOutputStream oos=new ObjectOutputStream(gmSocket.getOutputStream());
    ObjectInputStream ois = new ObjectInputStream(gmSocket.getInputStream());
    protocols.Message msg=new protocols.join.JoinRequest(min,max,getPort());
    oos.writeObject(msg);
    oos.flush();
    msg = (protocols.Message) ois.readObject();
    if (!(msg instanceof protocols.join.JoinReply))
    {
        throw new RuntimeException("Group Manager did not respond properly.");
    }
    protocols.join.JoinReply jr=(protocols.join.JoinReply)msg;
    if (!(jr.success))
    {
        throw new RuntimeException(jr.errorMessage);
    }
    log("Joined the Group!");
    gmSocket.close();
    // fire Joined Group Event !
    ChangeEvent e;
    e=new JoinedGroupEvent(this,jr.groupSize);
    fireChangeEvent(e);
    myIpap=jr.ipap;
    java.util.Enumeration enum=jr.userRecords.elements();
    while (enum.hasMoreElements()) addMember((groupmanager.ClientRecord)enum.
        nextElement());
    if (gmcm!=null) gmcm.shutdown();
    gmcm=new GMConnectionManager(this,GMAddr);
    gmcm.start();
```

```
}


public void leave(IpAndPort GMAddr) throws Exception
{
    log("Leaving Group...");
    Socket gmSocket=new Socket();
    gmSocket.setTcpNoDelay(true);
    gmSocket.setSoTimeout(10000);
    gmSocket.connect(new InetSocketAddress(GMAddr.getAddr(),GMAddr.getPort())
        ,10000);
    ObjectOutputStream oos=new ObjectOutputStream(gmSocket.getOutputStream());
    protocols.Message msg=new protocols.join.LeaveRequest(getPort());
    oos.writeObject(msg);
    oos.flush();
    gmSocket.close();
    log("Left the Group.");
}

private void addMember(groupmanager.ClientRecord cr)
{
    int index=getPtListIndex(cr.ipap);
    if (index!=-1) return;
    pinger.PingerThread pt=new pinger.PingerThread(this,cr.ipap);
    ptList.addElement(pt);
    if (pingingEnabled) pt.setSuspended(false);
    ChangeEvent e=new UpdateEvent(this,cr,true,"Joined the group.");
    fireChangeEvent(e);
    log(cr.ipap+" joined the group!");
}

private void removeMember(groupmanager.ClientRecord cr, String reason)
{
    int index=getPtListIndex(cr.ipap);
    if (index==-1) return;
    pinger.PingerThread pt=(pinger.PingerThread)ptList.elementAt(index);
    ptList.removeElement(pt);
    ChangeEvent e=new UpdateEvent(this,cr, false, reason);
    fireChangeEvent(e);
    pt.shutdown();
    log(pt.getIpAndPort()+" left the group! ("+reason+")");
}

protected void updateMembers(java.util.Vector clientRecords)
{
    java.util.Enumeration enum=clientRecords.elements();
    groupmanager.ClientRecord cr;
    IpAndPort ipap;
    int index;
    while (enum.hasMoreElements())
    {
        cr=((groupmanager.ClientRecord)enum.nextElement());
        index=getPtListIndex(cr.ipap);
        if (index==-1)
        {
            addMember(cr);
        }
    }

    enum=ptList.elements();
    while (enum.hasMoreElements())
    {
        ipap=((pinger.PingerThread)enum.nextElement()).getIpAndPort();
        index=getClientRecordIndex(clientRecords,ipap);
        if (index==-1)
        {
```

```
            removeMember(new groupmanager.ClientRecord(ipap,0,0),"not included in
                update.");
        }
    }

    // check for duplicates now
    enum=ptList.elements();
    while (enum.hasMoreElements())
    {
        ipap=((pinger.PingerThread)enum.nextElement()).getIpAndPort();
        index=countOccurences(ipap);
        if (index>1) removeMember(new groupmanager.ClientRecord(ipap,0,0),"
            duplicate.");
    }

    try
    {
        if (getPtListIndex(myIpap)==-1)
        {
            log("Fatal Error: Could not find myself ("+myIpap+") in the group.");
            shutdown();
            return;
        }
    } catch (Exception e)
    {
        log("Error during internal update: "+e.getMessage());
    }
    enum=ptList.elements();
    while (enum.hasMoreElements())
    {
        ipap=((pinger.PingerThread)enum.nextElement()).getIpAndPort();
        System.out.println("after updt: "+ipap);
    }
    System.out.println("---");
}


private int countOccurences(IpAndPort ipap)
{
    java.util.Enumeration enum=ptList.elements();
    int i=0;
    while (enum.hasMoreElements())
    {
        if (((pinger.PingerThread)enum.nextElement()).getIpAndPort().toString().
            equals(ipap.toString())) i++;
    }
    return i;
}


private int getClientRecordIndex(java.util.Vector clientRecords, IpAndPort ipap)
{
    java.util.Enumeration enum=clientRecords.elements();
    int i=0;
    while (enum.hasMoreElements())
    {
        if (((groupmanager.ClientRecord)enum.nextElement()).ipap.toString().
            equals(ipap.toString())) return i;
        i++;
    }
    return -1;
}


private int getPtListIndex(IpAndPort ipap)
{
```

```
    java.util.Enumeration enum=ptList.elements();
    int i=0;
    while (enum.hasMoreElements())
    {
        if (((pinger.PingerThread)enum.nextElement()).getIpAndPort().toString().
            equals(ipap.toString())) return i;
        i++;
    }
    return -1;
}



/** This method is called from the Servant in order for the Client Daemon to
    obtain the credential
through the GM. Due to bugs (streamcorruptedexceptions) we need to pass the
    references of the
object(in/out)put streams as well. */
protected void obtainCredential(Socket gmSocket, ObjectOutputStream oos)
{
    try
    {
        if (gmcm!=null) gmcm.shutdown();
        log("giving positive reply to group manager...");
        protocols.Message msg=new protocols.MessageBinary(true,null);
        oos.writeObject(msg);
        oos.flush();

        // now Group Manager should have forwarded this socket to Issuer
        // so we need to write again the objectoutputstream header
        log("asking for credential...");

        oos=new ObjectOutputStream(gmSocket.getOutputStream()); //this writes
            header to socket!
        msg=new protocols.issuing.IssueRequest();
        oos.writeObject(msg);
        oos.flush();


        // read response from Issuer

        ObjectInputStream ois=new ObjectInputStream(gmSocket.getInputStream());
        msg = (protocols.Message) ois.readObject();

        if (msg instanceof protocols.issuing.IssueReply)
        {
            protocols.issuing.IssueReply ir=(protocols.issuing.IssueReply)msg;
        } else
        {
            throw new RuntimeException("Unrecognised message from "+gmSocket.
                getInetAddress().getHostAddress()+":"+gmSocket.getPort());
        }
        log("Job Finished (todo: obtain receipt!)");
        gmSocket.close();
    } catch (Exception e)
    {
        e.printStackTrace();
        log("Error while obtaining credential: "+e.getMessage());
    }
    shutdown();
}


/**
  *   This method has to be called in order for the Client Daemon to start.
```

```
  */
public final void run()
{
    isActive=true;

    log(NAME+" is running.");
    while (isActive)
    {
        try
        {
            new Servant(this, ss.accept()); // this should accept pinging
                requests, (maybe also requests from the server?).
        } catch (Exception e)
        {
            e.printStackTrace();
            log("Client exception: "+e.getMessage());
        }
    }
}

/**
  * As the stop method in {@link Thread} is deprecated, this method should be
      called in order
  * to properly stop a running Discovery Server.
  */
public final void shutdown()
{
    if (!isActive) return;
    isActive=false;
    log("Attempting to stop "+NAME+"...");
    if (gmcm!=null) gmcm.shutdown();
    java.util.Enumeration enum=ptList.elements();
    while (enum.hasMoreElements())
    {
        pinger.PingerThread pt=(pinger.PingerThread)enum.nextElement();
        pt.shutdown();
    }
    try
    {
        ss.close();
        join();
    } catch (Exception e)
    {
        log("Error while closing: "+e.getMessage());
    }
    log(NAME+" stopped.");
}

public void stateChanged(pinger.PingerThread pt, boolean fatal)
{
    ChangeEvent e=new ClientConnectionChangeEvent(this,pt.getIpAndPort(), pt.
        getCurrentState(), fatal);
    fireChangeEvent(e);
}



public void stateChanged(boolean joining, String errorMessage, groupmanager.
    ClientRecord cr)
{
    if (joining)
    {
        addMember(cr);
    } else
    {
```

```
            removeMember(cr,errorMessage);
        }
    }



    public final void log(String s){log.println("["+df.format(new java.util.Date())+"
        /"+getTime()+"] "+s);log.flush();}

    protected final static String readInputStream(InputStream is) throws IOException
    {
        byte[] reply = new byte[65536];
        int size=is.read(reply);
        if (size>0) return new String(reply, 0, size); else return null;
    }

    public void addChangeListener(ChangeListener l)
    {
        listeners.add(ChangeListener.class, l);
    }

    public void removeChangeListener(ChangeListener l)
    {
        listeners.remove(ChangeListener.class, l);
    }

    public void setPingingEnabled(boolean set)
    {
        pingingEnabled=set;
        java.util.Enumeration enum=ptList.elements();
        while (enum.hasMoreElements())
        {
            pinger.PingerThread pd=(pinger.PingerThread)enum.nextElement();
            pd.setSuspended(!set);
        }
    }

    public boolean getPingingEnabled()
    {
        return pingingEnabled;
    }
    protected void fireChangeEvent(ChangeEvent e)
    {
        Object[] array = listeners.getListenerList();
        for (int i = array.length-2; i>=0; i-=2)
        {
            if (array[i]==ChangeListener.class)
            {
                ((ChangeListener)array[i+1]).stateChanged(e);
            }
        }
    }

    public int getPort()
    {
        return ss.getLocalPort();
    }

    public long getTime()
    {
        return new java.util.Date().getTime();
    }


}
```

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package client;

import protocols.IpAndPort;

/**
  *   This class is implements an event that allows the Client GUI to update itself.
  *
  *   @author Andreas Pashalidis
  */

public class ClientConnectionChangeEvent extends javax.swing.event.ChangeEvent
{
    private boolean fatal;
    private IpAndPort client;
    private String state;

    protected ClientConnectionChangeEvent(Object source, IpAndPort client, String
        state, boolean fatal)
    {
        super(source);
        this.client=client;
        this.fatal=fatal;
        this.state=state;
    }

    public boolean isFatal()
    {
        return fatal;
    }

    public IpAndPort getClientIpap()
    {
        return client;
    }

    public String getState()
    {
        return state;
    }


}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
```

```
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package client;


/**
  *   This class is implements an event that relates to a Group Manager.
  *
  *   @author Andreas Pashalidis
  */

public class FatalEvent extends javax.swing.event.ChangeEvent
{
    private int index;
    private String state;
    private boolean fatal;

    protected FatalEvent(Object source, int index, String state, boolean fatal)
    {
        super(source);
        this.index=index;
        this.state=state;
        this.fatal=fatal;
    }

    protected void setIndex(int index)
    {
        this.index=index;
    }

    public int getIndex()
    {
        return index;
    }

    public String getState()
    {
        return state;
    }

    public boolean getFatal()
    {
        return fatal;
    }
}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
```

```
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package client;

import java.net.*;
import java.io.*;
import protocols.IpAndPort;
import javax.swing.event.EventListenerList;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;

/**
  *   This class implements enables a user to ping another one, all the time.
  *   It registers the Group Manager with the Discovery Server, it maintains the
      connection, and it
  *   deregisters when appropriate.
  *   @author Andreas Pashalidis
  */

public class GMConnectionManager extends Thread
{
    public Client cl;
    public IpAndPort ipap;
    public volatile boolean isActive;


    public GMConnectionManager(Client cl, IpAndPort GMAddr)
    {
        this.cl=cl;
        this.ipap=GMAddr;
        isActive=false;
    }


    public void run()
    {
        isActive=true;
        while (isActive)
        {
            try
            {
                sleep(20000);
                Socket memberSocket=new Socket();
                memberSocket.setKeepAlive(true);
                memberSocket.setTcpNoDelay(true);
                memberSocket.setSoTimeout(25000);
                memberSocket.connect(new InetSocketAddress(ipap.getAddr(),ipap.
                    getPort()),10000);
                ObjectOutputStream oos=new ObjectOutputStream(memberSocket.
                    getOutputStream());
                ObjectInputStream ois = new ObjectInputStream(memberSocket.
                    getInputStream());
                protocols.Message msg=new protocols.update.UpdateRequest();
                oos.writeObject(msg);
                oos.flush();
                msg = (protocols.Message) ois.readObject();
                if (!(msg instanceof protocols.update.UpdateReply))
                {
                    throw new RuntimeException("response not recognised.");
```

```
                }
                cl.updateMembers(((protocols.update.UpdateReply)msg).clientRecords);
                memberSocket.close();
            } catch(Exception ex)
            {
                ex.printStackTrace();
                cl.log("Error occurred while requesting group update: "+ex.getMessage
                    ());
            }
        }
    }

    public final void shutdown()
    {
        isActive=false;
        cl.log("Stopping periodic update service...");
    }
}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package client;


/**
  *   This class is implements an event that allows the Client GUI to update itself.
  *
  *   @author Andreas Pashalidis
  */

public class JoinedGroupEvent extends javax.swing.event.ChangeEvent
{
    private int groupSize;

    protected JoinedGroupEvent(Object source, int groupSize)
    {
        super(source);
        this.groupSize=groupSize;
    }

    public int getGroupSize()
    {
        return groupSize;
    }
}

/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis
```

```
    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package client;

import java.net.*;
import java.io.*;
import java.util.Vector;
import protocols.Message;
import protocols.register.*;

final class Servant extends Thread
{
    private static long total=0;
    private long id;
    private Socket peer;
    private Client cl;

    protected Servant(Client cl, Socket peer)
    {
        this.cl=cl;
        this.peer=peer;
        total++;
        id=total;
        start();
    }

    public void run()
    {
        String peerString=peer.getInetAddress().getHostName()+":"+peer.getPort();
        try
        {
            ObjectOutputStream oos=new ObjectOutputStream(peer.getOutputStream());
            ObjectInputStream ois = new ObjectInputStream(peer.getInputStream());
            protocols.Message msg = (protocols.Message) ois.readObject();
            if (msg instanceof protocols.echo.MessageEcho)
            {
                if (((protocols.echo.MessageEcho)msg).echo)
                {
                    throw new RuntimeException("echo request from "+peerString+"
                        without echo bit set.");
                }
//                log("serving echo request from "+peerString);
                ((protocols.echo.MessageEcho)msg).echo=true;
                oos.writeObject(msg);
                oos.flush();
            } else
            if (msg instanceof protocols.update.SingleUpdate)
            {
                protocols.update.SingleUpdate su=(protocols.update.SingleUpdate)msg;
                if (su.joining)
                {
                    cl.log("Update received: "+su.cr.ipap+" joined!");
```

```
                } else
                {
                    cl.log("Update received: "+su.cr.ipap+" left.");
                }
                cl.stateChanged(su.joining, su.errorMessage, su.cr);
            } else
            if (msg instanceof protocols.run.RunRequest)
            {
                cl.log("Serving istruction to obtain credential...");
                cl.obtainCredential(peer,oos);
                return;
            } else
            {
                throw new RuntimeException("unrecognised request from "+peerString);
            }
            peer.close();
        } catch (Exception e)
        {
            e.printStackTrace();
            log("Serving Error: "+peerString+" "+e.getMessage()+" ("+id+").");
        }
    }


    private void log(String s){cl.log(id+" - "+s);}


}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package client;


/**
  *   This class is implements an event that allows the Client GUI to update itself.
  *
  *   @author Andreas Pashalidis
  */

public class UpdateEvent extends javax.swing.event.ChangeEvent
{
    private groupmanager.ClientRecord cr;
    private boolean isPresent;
    private String comment;

    protected UpdateEvent(Object source, groupmanager.ClientRecord cr, boolean
        isPresent, String comment)
    {
        super(source);
```

```
        this.cr=cr;
        this.isPresent=isPresent;
        this.comment=comment;
    }

    public groupmanager.ClientRecord getClientRecord()
    {
        return cr;
    }

    public boolean present()
    {
        return isPresent;
    }
    public String getMessage()
    {
        return comment;
    }
}
```

B.3 Discovery Server

This section provides the discovery server source code.

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package discoveryserver;


import java.net.*;
import java.io.*;
import protocols.IpAndPort;

/**
  *   This class implements a simple HTTP proxy daemon with Single Sign-On
      functionality into websites. If it
  *   bla bla bla
  *   the daemon, the <code>shutdown</code> method should be used.
  *
  *   @author       Andreas Pashalidis
  */

public final class DiscoveryServer extends Thread
{
```

```
protected static final String NAME="Discovery Server v.0.2";
private static final String CONSOLENOTICE="\n           Discovery Server -
    written by Andreas Pashalidis\n           
    ------------------------------------\nThis software is provided \"as is\"
    without warranty of any kind.\nNo responsibility will be accepted for any
    negative effects the\nsoftware may cause. You use it on your own risk.\n("+
    NAME+")\n";
private static final java.text.DateFormat df = java.text.DateFormat.
    getDateTimeInstance(3,2);

private PrintWriter log;
private ServerSocket ss;
private boolean isActive;
private java.util.Vector GMRecords;


/** Creates a Disvocery Server that will run on port 8080. Log messages will be
    sent to standard output.*/
public DiscoveryServer() throws Exception
{
    this(8080, new PrintWriter(System.out, true));
}

/** Creates a Discovery Server that will run on the specified port. Log messages
    will be sent to standard output.*/
public DiscoveryServer(int port) throws Exception
{
    this(port, new PrintWriter(System.out, true));
}

/**
  *   This is the full constructor that provides the maximum flexibility and
      functionality. It creates a Discovery Server
  *   that will run on the specified port and sent log messages to the specified
      {@link PrintWriter}.
  *
  *   @param port       the port the Impostor proxy shall run on
  *   @param log        the {@link PrintWriter} log messages shall be sent to
  */
public DiscoveryServer(int port, PrintWriter log) throws Exception
{
    System.out.println(CONSOLENOTICE);
    if (port<1 || port>65535) throw new IllegalArgumentException("Not a valid
        port number: "+port);
    if (log==null) throw new NullPointerException("Logger is null!");
    this.log=log;
    setDaemon(true);
    log("Attempting to start "+NAME+" on port "+port+"...");
    ss=new ServerSocket(port);
    isActive=false;
    GMRecords=new java.util.Vector();
    log(NAME+" is ready...");
}

/**
  *   This method has to be called in order for the Discovery Server to start.
  */

public final void run()
{
    isActive=true;
    log(NAME+" is running.");
    while (isActive)
    {
        try
        {
```

```
                new Servant(this, ss.accept());
            } catch (IOException e)
            {
                e.printStackTrace();
                log("Fatal I/O Exception: "+e.getMessage());
            }
        }
    }

    /**
      * As the stop method in {@link Thread} is deprecated, this method should be
        called in order
      * to properly stop a running Discovery Server.
      */
    public final void shutdown() throws Exception
    {
        if (!isActive) return;
        log("Attempting to stop "+NAME+" ...");
        ss.close();
        isActive=false;
        join();
        log(NAME+" stopped.");
    }


    protected boolean addGMRecord(GMRecord gmr)
    {
        java.util.Enumeration enum=GMRecords.elements();
        String addr;
        while (enum.hasMoreElements())
        {
            addr=(((GMRecord)enum.nextElement()).getIpAndPort()).getAddr().
                getHostAddress();
//            if (addr.equals(gmr.getIpAndPort().getAddr().getHostAddress())) return
    false;
        }
        GMRecords.addElement(gmr);
        return true;
    }

    protected java.util.Vector getGMRecords(int low, int high)
    {
        java.util.Vector results=new java.util.Vector();
        for (int i=0; i<GMRecords.size(); i++)
        {
            if (((GMRecord)GMRecords.elementAt(i)).isCompatible(low,high)) results.
                addElement(GMRecords.elementAt(i));
        }
        return results;
    }
    protected final void log(String s){log.println("["+df.format(new java.util.Date()
        )+"/"+getTime()+"] "+s);log.flush();}

    protected final static String readInputStream(InputStream is) throws IOException
    {
        byte[] reply = new byte[65536];
        int size=is.read(reply);
        if (size>0) return new String(reply, 0, size); else return null;
    }

    protected long getTime()
    {
        return new java.util.Date().getTime();
    }
}
```

```
/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package discoveryserver;

import protocols.IpAndPort;

public final class GMRecord implements java.io.Serializable
{
    private IpAndPort ipap;
    private int groupSize;

    protected GMRecord(IpAndPort ipap, int groupSize)
    {
        this.ipap=ipap;
        this.groupSize=groupSize;
    }

    protected boolean isCompatible(int low, int high)
    {
        if (groupSize<low) return false;
        if (groupSize>high) return false;
        return true;
    }

    public IpAndPort getIpAndPort()
    {
        return ipap;
    }
    public int getGroupSize()
    {
        return groupSize;
    }
}

/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.
```

```
    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package discoveryserver;

import java.net.*;
import java.io.*;
import java.util.Vector;
import protocols.Message;
import protocols.register.*;

final class Servant extends Thread
{
    private static long total=0;
    private long id;
    private Socket client;
    private DiscoveryServer ds;
    private boolean isActive;

    protected Servant(DiscoveryServer ds, Socket client)
    {
        this.ds=ds;
        this.client=client;
        total++;
        id=total;
        start();
    }

    public void run()
    {
        isActive=true;
        String clientString=client.getInetAddress().getHostName()+":"+client.getPort
            ();
        try
        {
            // get the client socket's streams
            ObjectInputStream ois = new ObjectInputStream(client.getInputStream());
            ObjectOutputStream oos=new ObjectOutputStream(client.getOutputStream());
            protocols.Message msg = (protocols.Message) ois.readObject();

            // we are expecting either a group manager to register with the discovery
                service
            // or
            // a user to issue a discovery request.

            if (msg instanceof protocols.discover.DiscoveryRequest) // it is a user
            {
                int low=((protocols.discover.DiscoveryRequest)msg).low;
                int high=((protocols.discover.DiscoveryRequest)msg).high;
                log("Serving Discovery Request from "+clientString+" for Group Size "
                    +low+"-"+high+".");
                Vector GMRecords=ds.getGMRecords(low,high);
                msg=new protocols.discover.DiscoveryReply(GMRecords);
                oos.writeObject(msg);
                oos.flush();
                client.close();
                return;
            }
            else if (msg instanceof protocols.register.RegisterRequest) // it is a
                group manager
            {
                log("Serving Registration Request from "+clientString+" for Group
                    Size "+((protocols.register.RegisterRequest)msg).groupSize+".");
```

```
                // registering group manager with dicovery server daemon
                protocols.IpAndPort ipap=new protocols.IpAndPort(client.
                    getInetAddress(),((protocols.register.RegisterRequest)msg).port);
                GMRecord gmr=new GMRecord(ipap,((protocols.register.RegisterRequest)
                    msg).groupSize);
                if (!(ds.addGMRecord(gmr)))
                    msg=new protocols.MessageBinary(false,"registration refused.");
                else
                    msg=new protocols.MessageBinary(true,null);
                oos.writeObject(msg);
                oos.flush();
            }
            else if (msg instanceof protocols.echo.MessageEcho)
            {
                if (((protocols.echo.MessageEcho)msg).echo)
                {
                    throw new RuntimeException("echo request from "+clientString+"
                        without echo bit set.");
                }
                log("Serving Echo Request from "+clientString);
                ((protocols.echo.MessageEcho)msg).echo=true;
                oos.writeObject(msg);
                oos.flush();
            } else
            {
                throw new Exception("Unrecognised message from "+clientString);
            }
            client.close();
        } catch (Exception e)
        {
            e.printStackTrace();
            log("Exception: "+clientString+" "+e.getMessage()+" ("+id+").");
            return;
        }
    }


    private void log(String s){ds.log(id+" - "+s);}


}
```

B.4 Group Manager

This section provides the discovery server source code.

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
```

```
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;

import protocols.IpAndPort;

/**
  *   This class is implements an event that allows the Client GUI to update itself.
  *
  *   @author Andreas Pashalidis
  */

public class ClientConnectionChangeEvent extends javax.swing.event.ChangeEvent
{
    private boolean fatal;
    private IpAndPort client;
    private String state;

    protected ClientConnectionChangeEvent(Object source, IpAndPort client, String
        state, boolean fatal)
    {
        super(source);
        this.client=client;
        this.fatal=fatal;
        this.state=state;
    }

    public boolean isFatal()
    {
        return fatal;
    }

    public IpAndPort getClientIpap()
    {
        return client;
    }

    public String getState()
    {
        return state;
    }


}

/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/
```

```
package groupmanager;

import java.util.Vector;
import protocols.IpAndPort;

public final class ClientRecord implements java.io.Serializable
{
    public IpAndPort ipap;
    public int min, max;

    public ClientRecord(IpAndPort ipap, int min, int max)
    {
        this.ipap=ipap;
        this.min=min;
        this.max=max;
    }
}

/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;

import java.net.*;
import java.io.*;
import java.util.Vector;
import protocols.*;

final class CredentialIssueProxy extends Thread
{
    private static long total=0;
    private long id;

    private GroupManager gm;
    IpAndPort ipap;

    protected CredentialIssueProxy(GroupManager gm, IpAndPort ipap)
    {
        this.gm=gm;
        this.ipap=ipap;
        total++;
        id=total;
    }

    public void run()
    {
        try
        {
```

```
            Socket memberSocket=new Socket();
            memberSocket.setKeepAlive(true);
            memberSocket.setTcpNoDelay(true);
            memberSocket.setSoTimeout(25000);
            memberSocket.connect(new InetSocketAddress(ipap.getAddr(),ipap.getPort())
                ,10000);
            ObjectOutputStream oos=new ObjectOutputStream(memberSocket.
                getOutputStream());
            protocols.Message msg=new protocols.run.RunRequest();
            oos.writeObject(msg);
            oos.flush();

            ObjectInputStream ois = new ObjectInputStream(memberSocket.getInputStream
                ());
            msg = (protocols.Message) ois.readObject();
            if (!(msg instanceof protocols.MessageBinary))
            {
                throw new RuntimeException("Credential Issue Proxy: Unrecognized
                    reply from "+ipap);
            }
            protocols.MessageBinary mb=(protocols.MessageBinary)msg;
            if (!(mb.success))
            {
                throw new RuntimeException("Credential Issue Proxy: "+mb.errorMessage
                    );
            }
            Socket issuerSocket=new Socket();
            issuerSocket.connect(new InetSocketAddress("134.219.23.130",1863),10000);

            TcpForwarder clientToIssuer=new TcpForwarder(gm, memberSocket,
                issuerSocket);
            TcpForwarder issuerToClient=new TcpForwarder(gm, issuerSocket,
                memberSocket);

            gm.log("Acting as proxy for "+ipap);
            clientToIssuer.start();
            issuerToClient.start();

            while(clientToIssuer.isAlive() || issuerToClient.isAlive())
            {
                sleep(350);
            }

            issuerSocket.close();
            memberSocket.close();
        } catch (Exception e)
        {
            e.printStackTrace();
            gm.log("Error while acting as proxy for "+ipap+": "+e.getMessage()+" ("+
                id+")");
        }
        gm.log("Closing down...");
        try
        {
            gm.shutdown();
        } catch (Exception e)
        {
            gm.log("Error while shutting down Group Manager: "+e.getMessage());
        }
    }


}

/*
    Copyright (C) 2003 Andreas Pashalidis
```

```
    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;


import java.net.*;
import java.io.*;
import protocols.IpAndPort;
import javax.swing.event.EventListenerList;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;

/**
  *   This class manages the communication between the Group Manager daemon and one
      Discovery Server.
  *   It registers the Group Manager with the Discovery Server, it maintains the
      connection, and it
  *   deregisters when appropriate.
  *   @author Andreas Pashalidis
  */

public final class DSConnectionManager extends pinger.PingerThread
{
    protected DSConnectionManager(GroupManager parent, IpAndPort dsAddr)
    {
        super(parent, dsAddr);
    }

    protected void register()
    {
        try
        {
            Socket dsSocket=new Socket();
            dsSocket.setKeepAlive(true);
            dsSocket.setTcpNoDelay(true);
            dsSocket.setSoTimeout(10000);
            state="Connecting...";
            pd.log("Connecting to "+ipap+"...");
            pd.stateChanged(this, false);
            dsSocket.connect(new InetSocketAddress(ipap.getAddr(),ipap.getPort())
                ,10000);
            state="Registering...";
            pd.log("Registering with "+ipap+"...");
            pd.stateChanged(this, false);
            protocols.Message msg=new protocols.register.RegisterRequest(((
                GroupManager)pd).getGroupSize(),((GroupManager)pd).getPort());
            ObjectOutputStream oos=new ObjectOutputStream(dsSocket.getOutputStream())
                ;
            oos.writeObject(msg);
            oos.flush();

            ObjectInputStream ois = new ObjectInputStream(dsSocket.getInputStream());
```

```
            msg = (protocols.Message) ois.readObject();
            if (!(msg instanceof protocols.MessageBinary))
            {
                throw new RuntimeException("Server did not respond properly.");
            }
            if (!((protocols.MessageBinary)msg).success)
            {
                throw new RuntimeException(((protocols.MessageBinary)msg).
                    errorMessage);
            }
            dsSocket.close();
            state="Registered.";
            pd.log("Registered with "+ipap+" ...");
            pd.stateChanged(this, false);
        } catch(Exception ex)
        {
            pd.log("Exception: "+ex.getMessage());
            state=ex.getMessage();
            pd.stateChanged(this, true);
        }
    }


}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;


/**
  *   This class is implements an event that relates to a Discovery Server.
  *
  *   @author Andreas Pashalidis
  */

public class DSEvent extends javax.swing.event.ChangeEvent
{
    private int index;
    private String state;
    private boolean fatal;

    protected DSEvent(Object source, int index, String state, boolean fatal)
    {
        super(source);
        this.index=index;
        this.state=state;
        this.fatal=fatal;
    }
```

```
    protected void setIndex(int index)
    {
        this.index=index;
    }

    public int getIndex()
    {
        return index;
    }

    public String getState()
    {
        return state;
    }

    public boolean getFatal()
    {
        return fatal;
    }
}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;


import java.net.*;
import java.io.*;
import protocols.IpAndPort;
import javax.swing.event.EventListenerList;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;

/**
  *   This class implements the functionality of the Group Manager daemon.
  *
  *   @author Andreas Pashalidis
  */

public final class GroupManager extends Thread implements pinger.PingingDaemon
{
    protected static final String NAME="Group Manager v.0.2";
    private static final String CONSOLENOTICE="\n           Group Manager - written
        by Andreas Pashalidis\n           ----------------------------------------\
        nThis software is provided \"as is\" without warranty of any kind.\nNo
        responsibility will be accepted for any negative effects the\nsoftware may
        cause. You use it on your own risk.\n("+NAME+")\n";
    private static final java.text.DateFormat df = java.text.DateFormat.
        getDateTimeInstance(3,2);
    private EventListenerList listeners;
```

```
private PrintWriter log;
private ServerSocket ss;
private int groupSize;
private volatile java.util.Vector dsList, dscmList, clientRecords, clientPingers;
private volatile boolean isActive, acceptMoreMembers;

/** Creates a Group Manager that will log messages to the standard output.*/
public GroupManager(Init init) throws Exception
{
    this(init, new PrintWriter(System.out, true));
}


/**
  *   This is the full constructor that provides the maximum flexibility and
      functionality. It creates a Group Manager
  *   that will log messages to the specified {@link PrintWriter}.
  *
  *   @param init        the initialization object for this Group Manager instance
  *   @param log         the {@link PrintWriter} log messages shall be sent to
  */
public GroupManager(Init init, PrintWriter log) throws Exception
{
    listeners=new EventListenerList();
    clientRecords=new java.util.Vector();
    clientPingers=new java.util.Vector();
    System.out.println(CONSOLENOTICE);
    if (init.getPort()<1 || init.getPort()>65535) throw new
        IllegalArgumentException("Not a valid port number: "+init.getPort());
    if (log==null) throw new NullPointerException("Logger is null!");
    this.log=log;
    this.dsList=init.getDSList();
    if (this.dsList.size()<1) throw new IllegalArgumentException("No Discovery
        Servers Found in Initalization Object!");
    this.groupSize=init.getGroupSize();
    if (this.groupSize<2) throw new IllegalArgumentException("Not a valid group
        size: "+this.groupSize);

    acceptMoreMembers=true;
    setDaemon(true);
    log("Attempting to start "+NAME+" on port "+init.getPort()+"...");
    ss=new ServerSocket(init.getPort());
    log(NAME+" is ready.");
}

/**
  *   This method has to be called in order for the Discovery Server to start.
  */

public final void run()
{
    isActive=true;
    dscmList=new java.util.Vector();
    java.util.Enumeration enum=dsList.elements();
    log("Registering at Discovery Server(s)...");
    while (enum.hasMoreElements())
    {
        DSConnectionManager dscm=new DSConnectionManager(this,(IpAndPort)enum.
            nextElement());
        dscmList.addElement(dscm);
        dscm.register(); // this registers the GM with the discovery server
        dscm.setSuspended(false); // this starts pinging
    }
    log(NAME+" is running.");
    while (isActive)
```

```
        {
            try
            {
                new Servant(this, ss.accept(), acceptMoreMembers);
            } catch (Exception e)
            {
                e.printStackTrace();
                log("Exception: "+e.getMessage());
            }
        }
    }

    /**
      * As the stop method in {@link Thread} is deprecated, this method should be
        called in order
      * to properly stop a running Discovery Server.
      */
    public final void shutdown() throws Exception
    {
        if (!isActive) return;
        log("Attempting to stop "+NAME+"...");
        java.util.Enumeration enum=dsList.elements();
        while (enum.hasMoreElements())
        {
            ((DSConnectionManager)enum.nextElement()).shutdown();
        }
        ss.close();
        isActive=false;
        join();
        log(NAME+" stopped.");
    }



    public void stateChanged(pinger.PingerThread pt, boolean fatal)
    {
//        log("new state for "+pt.getIpAndPort()+": "+pt.getState());
        if (pt instanceof DSConnectionManager)
        {
            int index=dscmList.indexOf(pt);
            ChangeEvent e=new DSEvent(this,index,pt.getCurrentState(), fatal);
            fireChangeEvent(e);
            // connection with Discovery Server is Lost!
            // should implement a check here: if no alive DS servers left, all
                members should be notified!
            return;
        }
        // pt must be a client pinger
        if (fatal) // we lost a member, now we must tell all remaining members
        {
            removeMember(new groupmanager.ClientRecord(pt.getIpAndPort(),0,0), pt.
                getCurrentState());
        } else
        {
            ChangeEvent e=new ClientConnectionChangeEvent(this,pt.getIpAndPort(),pt.
                getCurrentState(), false);
            fireChangeEvent(e);
        }
    }


   protected void removeMember(groupmanager.ClientRecord cr, String reason)
    {
        int index=getClientIndex(cr.ipap);
        if (index==-1) return;
```

```
    // first remove the (ex-)member
    pinger.PingerThread pt=(pinger.PingerThread)clientPingers.elementAt(index);
    clientPingers.removeElementAt(index);
    clientRecords.removeElementAt(index);
    log("Removing "+pt.getIpAndPort()+" from the group! ("+reason+")");
    ChangeEvent e=new UpdateEvent(this,cr, false, reason);
    fireChangeEvent(e);
    // then update remaining members
    updateMembers(false,reason,cr);
    pt.shutdown();
}

protected boolean addMember(ClientRecord cr)
{
    int index=getClientIndex(cr.ipap);
    if (index!=-1) return false;
    // new member is about to join this group -- we must tell all existing
        members!
    updateMembers(true,null,cr);
    // now actually add the client...
    log("Adding "+cr.ipap+" ("+cr.min+","+cr.max+") to the group.");
    ChangeEvent e=new UpdateEvent(this,cr,true,"Just joined!");
    fireChangeEvent(e);
    pinger.PingerThread pt=new pinger.PingerThread(this,cr.ipap);
    clientRecords.addElement(cr);
    clientPingers.addElement(pt);
    pt.setSuspended(false);
    if (clientRecords.size()==groupSize) initiateProcedure();
    return true;
}


private void initiateProcedure()
{
    int[]  sorted=new int[groupSize];
    long[] delay=new long[groupSize];
    int t,j,i=0;
    java.util.Enumeration enum=clientPingers.elements();
    while (enum.hasMoreElements())
    {
        sorted[i]=i;
        delay[i++]=((pinger.PingerThread)(enum.nextElement())).getAverage();
    }

    for (i=0;i<groupSize;i++)
    System.out.println("BEFORE..."+i+":"+delay[i]+" rank:"+sorted[i]);

    long temp;
    for (i=0; i<groupSize-1; i++)
    {
        for (j=0; j<groupSize-1-i; j++)
        {
            if (delay[j+1]<delay[j])
            {
                temp=delay[j];
                delay[j]=delay[j+1];
                delay[j+1]=temp;
                t=sorted[j];
                sorted[j]=sorted[j+1];
                sorted[j+1]=t;
            }
        }
    }

    for (i=0;i<groupSize;i++)
    System.out.println("AFTER..."+i+":"+delay[i]+" rank:"+sorted[i]);
```

```
    IpAndPort ipap;
    for (i=0; i<groupSize; i++)
    {
        ipap=((ClientRecord)clientRecords.elementAt(sorted[i])).ipap;
        log("Starting proxy for "+ipap);
        CredentialIssueProxy cip=new CredentialIssueProxy(this,ipap);
        cip.start();
    }
}


private int getClientIndex(IpAndPort ipap)
{
    java.util.Enumeration enum=clientPingers.elements();
    int i=0;
    while (enum.hasMoreElements())
    {
        if (((pinger.PingerThread)enum.nextElement()).ipap.toString().equals(ipap
            .toString())) return i;
        i++;
    }
    return -1;
}


private void updateMembers(boolean joining, String errorMessage, ClientRecord cr)
{
    java.util.Enumeration enum=clientRecords.elements();
    IpAndPort ipap;
    while (enum.hasMoreElements())
    {
        ipap=((ClientRecord)enum.nextElement()).ipap;
        MemberUpdater mu=new MemberUpdater(this,ipap,joining,errorMessage,cr);
    }
}

protected java.util.Vector getClientRecords()
{
    return clientRecords;
}


public int getGroupSize()
{
    return groupSize;
}

public int getPort()
{
    return ss.getLocalPort();
}

public long getTime()
{
    return new java.util.Date().getTime();
}

public void addChangeListener(ChangeListener l)
{
    listeners.add(ChangeListener.class, l);
}

public void removeChangeListener(ChangeListener l)
{
    listeners.remove(ChangeListener.class, l);
```

```
    }

    protected void fireChangeEvent(ChangeEvent e)
    {
        Object[] array = listeners.getListenerList();
        for (int i = array.length-2; i>=0; i-=2)
        {
            if (array[i]==ChangeListener.class)
            {
                ((ChangeListener)array[i+1]).stateChanged(e);
            }
        }
    }
    public final void log(String s){log.println("["+df.format(new java.util.Date())+"
        /"+getTime()+"] "+s);log.flush();}

}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;


import java.util.Vector;
import protocols.IpAndPort;

/**
  *   This class implements an Initialisation Object for the Group Manager. It
      contains all the information to bootstrap a Group Manager deamon.
  *
  *   @author Andreas Pashalidis
  */

public final class Init
{
    private int port, groupSize;
    private Vector dsVector;


    public Init(int port, int groupSize)
    {
        this.port=port;
        this.groupSize=groupSize;
        dsVector=new java.util.Vector();
    }

    public void addDS(IpAndPort addr)
    {
        dsVector.addElement(addr);
    }
```

```
    protected Vector getDSList()
    {
        return dsVector;
    }

    protected int getPort()
    {
        return port;
    }

    protected int getGroupSize()
    {
        return groupSize;
    }
}

/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;

import java.net.*;
import java.io.*;
import java.util.Vector;
import protocols.*;

final class MemberUpdater extends Thread
{

    private GroupManager gm;
    private boolean joining;
    private String errorMessage;
    IpAndPort ipap;
    private ClientRecord cr;

    protected MemberUpdater(GroupManager gm, IpAndPort ipap, boolean joining, String
        errorMessage, ClientRecord cr)
    {
        this.gm=gm;
        this.ipap=ipap;
        this.joining=joining;
        this.errorMessage=errorMessage;
        this.cr=cr;
        start();
    }

    public void run()
    {
```

```
        protocols.Message msg=new protocols.update.SingleUpdate(joining,errorMessage,
            cr);
        try
        {
            Socket memberSocket=new Socket();
            memberSocket.setKeepAlive(true);
            memberSocket.setTcpNoDelay(true);
            memberSocket.setSoTimeout(50000);
            memberSocket.connect(new InetSocketAddress(ipap.getAddr(),ipap.getPort())
                ,10000);
            ObjectOutputStream oos=new ObjectOutputStream(memberSocket.
                getOutputStream());
            ObjectInputStream ois = new ObjectInputStream(memberSocket.getInputStream
                ());
            oos.writeObject(msg);
            oos.flush();
            memberSocket.close();
        } catch (Exception e)
        {
//            e.printStackTrace();
            gm.log("Error while trying to send update to "+ipap+": "+e.getMessage());
        }
    }


}

/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;

import java.net.*;
import java.io.*;
import java.util.Vector;
import protocols.*;

final class Servant extends Thread
{
    private static long total=0;
    private long id;
    private Socket client;
    private GroupManager gm;
    private boolean acceptMoreMembers;

    protected Servant(GroupManager gm, Socket client, boolean acceptMoreMembers)
    {
        this.client=client;
        this.gm=gm;
```

```
        this.acceptMoreMembers=acceptMoreMembers;
        total++;
        id=total;
        start();
    }

    public void run()
    {
        String clientString=client.getInetAddress().getHostName()+":"+client.getPort
            ();
        try
        {
            ObjectOutputStream oos=new ObjectOutputStream(client.getOutputStream());
            ObjectInputStream ois = new ObjectInputStream(client.getInputStream());
            protocols.Message msg = (protocols.Message) ois.readObject();
            if (msg instanceof protocols.join.JoinRequest)
            {
                protocols.join.JoinRequest jr=(protocols.join.JoinRequest)msg;
                int min=jr.min;
                int max=jr.max;
                int port=jr.port;
                log("Join group request from "+ clientString+"("+min+","+max+")");
                ClientRecord cr=null;
                if (!(acceptMoreMembers))
                {
                    log("Cannot accept "+clientString+" ("+min+","+max+"): group is
                        closed!");
                    msg=new protocols.join.JoinReply(false, "Group closed.", null,gm.
                        getGroupSize(),cr.ipap);
                } else
                if (gm.getGroupSize()<min)
                {
                    log(clientString+" ("+min+","+max+"): minimum too low!");
                    msg=new protocols.join.JoinReply(false, "Lower limit "+min+" too
                        low.", null,gm.getGroupSize(),cr.ipap);
                } else
                if (gm.getGroupSize()>max)
                {
                    log(clientString+" ("+min+","+max+"): maximum too high!");
                    msg=new protocols.join.JoinReply(false, "Upper limit "+max+" too
                        high.", null,gm.getGroupSize(),cr.ipap);
                } else
                {
                    cr=new ClientRecord(new IpAndPort(client.getInetAddress(),port),
                        min,max);
                    if (!(gm.addMember(cr)))
                    {
                        log(clientString+" ("+min+","+max+"): could not join the
                            group!");
                        msg=new protocols.join.JoinReply(false, "Group join denied.",
                             null,gm.getGroupSize(),cr.ipap);
                    } else
                    {
                        msg=new protocols.join.JoinReply(true,null,gm.
                            getClientRecords(),gm.getGroupSize(),cr.ipap);
                    }
                }
                oos.writeObject(msg);
                oos.flush();
            } else if (msg instanceof protocols.join.LeaveRequest)
            {
                protocols.join.LeaveRequest lr=(protocols.join.LeaveRequest)msg;
                log("Leave group request from "+ clientString);
                gm.removeMember(new ClientRecord(new IpAndPort(client.getInetAddress
                    (),lr.port),0,0),"leave requested");
            } else if (msg instanceof protocols.update.UpdateRequest)
```

```
            {
                protocols.update.UpdateRequest ur=(protocols.update.UpdateRequest)msg
                    ;
                log("Update request from "+ clientString);
                msg=new protocols.update.UpdateReply(gm.getClientRecords());
                oos.writeObject(msg);
                oos.flush();
            }
            else
            {
                log("unrecognised message from "+clientString);
            }
            client.close();
        } catch (Exception e)
        {
//              e.printStackTrace();
            log("Serving Error: "+clientString+" "+e.getMessage()+" ("+id+").");
            return;
        }
    }


    private void log(String s){gm.log(id+" - "+s);}


}

/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;

import java.net.*;
import java.io.*;
import java.util.Vector;
import protocols.*;

final class TcpForwarder extends Thread
{

    private GroupManager gm;
    private Socket from, to;

    protected TcpForwarder(GroupManager gm, Socket from, Socket to)
    {
        this.gm=gm;
        this.from=from;
        this.to=to;
    }
```

```
    public void run()
    {
        boolean isActive=true;
        byte[] buffer=new byte[32768];

        try
        {
            InputStream is  = from.getInputStream();
            OutputStream os = to.getOutputStream();
            int b;
            while (isActive)
            {
                b=is.read(buffer);
                if (b<0)
                {
                    isActive=false;
                    os.close();
                } else
                {
                    os.write(buffer,0,b);
                    os.flush();
                }
            }
        } catch (Exception e)
        {
            e.printStackTrace();
            gm.log("Error proxying from "+getIpap(from)+" to "+getIpap(to)+":"+e.
                getMessage());
        }
    }

    private static String getIpap(Socket s)
    {
        return s.getInetAddress().getHostAddress()+":"+s.getPort();
    }


}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package groupmanager;


/**
  *   This class is implements an event that allows the Client GUI to update itself.
  *
  *   @author Andreas Pashalidis
  */
```

```
public class UpdateEvent extends javax.swing.event.ChangeEvent
{
    private groupmanager.ClientRecord cr;
    private boolean isPresent;
    private String comment;

    protected UpdateEvent(Object source, groupmanager.ClientRecord cr, boolean
        isPresent, String comment)
    {
        super(source);
        this.cr=cr;
        this.isPresent=isPresent;
        this.comment=comment;
    }

    public groupmanager.ClientRecord getClientRecord()
    {
        return cr;
    }

    public boolean present()
    {
        return isPresent;
    }
    public String getMessage()
    {
        return comment;
    }
}
```

B.5 Issuer

This section provides the source code of the dummy issuer that was used to test the system.

```
/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package discoveryserver;

import protocols.IpAndPort;

public final class GMRecord implements java.io.Serializable
```

```
{
    private IpAndPort ipap;
    private int groupSize;

    protected GMRecord(IpAndPort ipap, int groupSize)
    {
        this.ipap=ipap;
        this.groupSize=groupSize;
    }

    protected boolean isCompatible(int low, int high)
    {
        if (groupSize<low) return false;
        if (groupSize>high) return false;
        return true;
    }

    public IpAndPort getIpAndPort()
    {
        return ipap;
    }
    public int getGroupSize()
    {
        return groupSize;
    }
}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package issuer;


import java.net.*;
import java.io.*;
import protocols.IpAndPort;

/**
  *   This class implements a simple HTTP proxy daemon with Single Sign-On
      functionality into websites. If it
  *   bla bla bla
  *   the daemon, the <code>shutdown</code> method should be used.
  *
  *   @author       Andreas Pashalidis
  */

public final class Issuer extends Thread
{
    protected static final String NAME="Credential␣Issuing␣Simulator␣v.0.2";
    private static final String CONSOLENOTICE="\n␣␣␣␣␣␣␣␣␣␣␣Credential␣Issuing␣Server
        ␣-␣written␣by␣Andreas␣Pashalidis\n␣␣␣␣␣␣␣␣␣␣␣
```

```
    ------------------------------------------\nThis software is provided \"as is\"
    without warranty of any kind.\nNo responsibility will be accepted for any
    negative effects the\nsoftware may cause. You use it on your own risk.\n("+
    NAME+")\n";
private static final java.text.DateFormat df = java.text.DateFormat.
    getDateTimeInstance(3,2);

private PrintWriter log;
private ServerSocket ss;
private boolean isActive;


/** Creates a Disvocery Server that will run on port 8080. Log messages will be
    sent to standard output.*/
public Issuer() throws Exception
{
    this(8080, new PrintWriter(System.out, true));
}

/** Creates a Credential Issuing Simulator Service that will run on the specified
     port. Log messages will be sent to standard output.*/
public Issuer(int port) throws Exception
{
    this(port, new PrintWriter(System.out, true));
}

/**
  *   This is the full constructor that provides the maximum flexibility and
      functionality. It creates a Discovery Server
  *   that will run on the specified port and sent log messages to the specified
      {@link PrintWriter}.
  *
  *   @param port       the port the Impostor proxy shall run on
  *   @param log        the {@link PrintWriter} log messages shall be sent to
  */
public Issuer(int port, PrintWriter log) throws Exception
{
    System.out.println(CONSOLENOTICE);
    if (port<1 || port>65535) throw new IllegalArgumentException("Not a valid
        port number: "+port);
    if (log==null) throw new NullPointerException("Logger is null!");
    this.log=log;
    setDaemon(true);
    log("Attempting to start "+NAME+" on port "+port+"...");
    ss=new ServerSocket(port);
    isActive=false;
    log(NAME+" is ready...");
}

/**
  *   This method has to be called in order for the Discovery Server to start.
  */

public final void run()
{
    isActive=true;
    log(NAME+" is running.");
    while (isActive)
    {
        try
        {
            new Servant(this, ss.accept());
        } catch (IOException e)
        {
            e.printStackTrace();
            log("Fatal I/O Exception: "+e.getMessage());
```

```
            }
        }
    }

    /**
      * As the stop method in {@link Thread} is deprecated, this method should be
          called in order
      * to properly stop a running Discovery Server.
      */
    public final void shutdown() throws Exception
    {
        if (!isActive) return;
        log("Attempting to stop "+NAME+" ...");
        ss.close();
        isActive=false;
        join();
        log(NAME+" stopped.");
    }


    protected final void log(String s){log.println("["+df.format(new java.util.Date()
        )+"/"+getTime()+"] "+s);log.flush();}

    protected final static String readInputStream(InputStream is) throws IOException
    {
        byte[] reply = new byte[65536];
        int size=is.read(reply);
        if (size>0) return new String(reply, 0, size); else return null;
    }

    protected long getTime()
    {
        return new java.util.Date().getTime();
    }
}

/*
    This class implements the Servant thread for a Discovery Server; it is a Thread

    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package issuer;

import java.net.*;
import java.io.*;
import java.util.Vector;
import protocols.Message;

final class Servant extends Thread
{
    private static long total=0;
```

```
    private long id;
    private Socket client;
    private Issuer issuer;
    private boolean isActive;

    protected Servant(Issuer issuer, Socket client)
    {
        this.issuer=issuer;
        this.client=client;
        total++;
        id=total;
        start();
    }

    public void run()
    {
        isActive=true;
        String clientString=client.getInetAddress().getHostName()+":"+client.getPort
            ();
        try
        {
            // get the client socket's streams
//            System.out.println("InputStream available: "+client.getInputStream().
    available());

            ObjectOutputStream oos=new ObjectOutputStream(client.getOutputStream());
            ObjectInputStream ois = new ObjectInputStream(client.getInputStream());
            protocols.Message msg = (protocols.Message) ois.readObject();

            if (msg instanceof protocols.issuing.IssueRequest) // it is a credential
                issuing request
            {
                protocols.issuing.IssueRequest ir=(protocols.issuing.IssueRequest)msg
                    ;
                log("Serving Credential Issuing Request from "+clientString+".");
                msg=new protocols.issuing.IssueReply();
                oos.writeObject(msg);
                oos.flush();
                log("Credential Issued to "+clientString);
            } else
            {
                throw new Exception("Unrecognised message from "+clientString);
            }
            client.close();
        } catch (Exception e)
        {
            e.printStackTrace();
            log("Exception: "+clientString+" "+e.getMessage()+" ("+id+").");
            return;
        }
    }


    private void log(String s){issuer.log(id+" - "+s);}


}
```

B.6 Other classes

This section provides the source code of other classes that do not directly belong to any of the above categories, or that are common to more than one of the above services.

```
/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package pinger;


import java.net.*;
import java.io.*;
import protocols.IpAndPort;
import javax.swing.event.EventListenerList;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;

/**
  *   This class implements enables a user to ping another one, all the time.
  *   It registers the Group Manager with the Discovery Server, it maintains the
      connection, and it
  *   deregisters when appropriate.
  *   @author Andreas Pashalidis
  */

public class PingerThread extends Thread
{
    public PingingDaemon pd;
    public IpAndPort ipap;
    private volatile boolean isActive, isSuspended;
    public String state="Not Initialized.";
    private long repetitions,average;

    public String getCurrentState()
    {
        return state;
    }

    public long getAverage()
    {
        return average;
    }

    public PingerThread(PingingDaemon pd, IpAndPort memberAddr)
    {
        this.pd=pd;
        this.ipap=memberAddr;
```

```
        isActive=false;
        isSuspended=true;
        repetitions=0;
        average=0;
        start();
    }


    public void run()
    {
        isActive=true;
        while (isActive)
        {
            try
            {
                while (isSuspended && isActive)
                {
                    sleep(350);
                }
                if (!isSuspended)
                {
//                    pd.log("...pinging peer "+ipap);
                    Socket memberSocket=new Socket();
                    memberSocket.setKeepAlive(true);
                    memberSocket.setTcpNoDelay(true);
                    memberSocket.setSoTimeout(25000);
                    memberSocket.connect(new InetSocketAddress(ipap.getAddr(),ipap.
                        getPort()),10000);
                    ObjectOutputStream oos=new ObjectOutputStream(memberSocket.
                        getOutputStream());
                    ObjectInputStream ois = new ObjectInputStream(memberSocket.
                        getInputStream());
                    protocols.Message msg=new protocols.echo.MessageEcho(false,pd.
                        getTime());
                    oos.writeObject(msg);
                    oos.flush();
                    msg = (protocols.Message) ois.readObject();
                    if (!(msg instanceof protocols.echo.MessageEcho))
                    {
                        throw new RuntimeException("response not recognised.");
                    }
                    if (!(((protocols.echo.MessageEcho)msg).echo))
                    {
                        throw new RuntimeException("echo bit not set.");
                    }
                    long latency=pd.getTime()-((protocols.echo.MessageEcho)msg).time;
                    repetitions++;
                    if (repetitions==1)
                    {
                        average=latency;
                    } else
                    {
                        average=(long)((average*.2)+(latency*.8));
                    }
                    state="latency (msec):"+repetitions+"/"+latency+"/"+average;
                    pd.stateChanged(this, false);
                    memberSocket.close();
                    sleep(5000);
                }
            } catch(Exception ex)
            {
//                ex.printStackTrace();
                pd.log("Pinger exception: "+ex.getMessage());
                state=ex.getMessage();
                pd.stateChanged(this, true);
                isSuspended=true;
```

```
            }
        }
    }


    public final void shutdown()
    {
        isActive=false;
        try
        {
            join();
        } catch (Exception e)
        {
            pd.log("Error while stopping pinger for "+ipap+": "+e.getMessage());
        }
        pd.log("Stopped pinger for peer "+ipap+" ...");
    }

    public final void setSuspended(boolean suspend)
    {
        if ((suspend)&&(!isSuspended))
        {
            state="last latency (msec): "+repetitions+"/"+average;
            pd.stateChanged(this, false);
        }
        isSuspended=suspend;
    }


    public IpAndPort getIpAndPort()
    {
        return ipap;
    }
}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

package pinger;


public interface PingingDaemon
{
    public void stateChanged(PingerThread pt, boolean fatal);
    public long getTime();
    public void log(String s);
}

/*
    Copyright (C) 2003 Andreas Pashalidis
```

```
    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;
import java.util.Vector;
import java.awt.Font;
import protocols.IpAndPort;

/**
  *   This class is a program that offers a user-friendly interface that controls a {
      @link groupmanager.GroupManager} daemon;
  */

public class SimpleClient implements WindowListener, KeyListener, MouseListener,
    ActionListener, javax.swing.event.ChangeListener
{
    private JFrame configFrame, workingFrame, currentFrame;
    private JTextField portField, minField, maxField, dsField, dsPortField; // for
        configFrame
    private JProgressBar progressBar; // for workingFrame
    private Vector clientRecords, clientStati, clientBoxes; // for working frame (
        clientRecords for housekeeping)
    Box     clientPanelBox;     // for workingFrame
    private JButton okButton;
    private JToggleButton pingButton;
    private client.Client cl;
    private IpAndPort GMAddr;

    public SimpleClient()
    {
        clientStati=new Vector();
        clientRecords=new Vector();
        clientBoxes=new Vector();
        //adopt to look-and-feel
        try
        {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e)
        {
            e.printStackTrace();
        }
        makeConfigFrame();
        makeWorkingFrame();
        currentFrame=configFrame;
        configFrame.pack();
        configFrame.show();

    }
```

```
public static void main(String[] argv)
{
    System.out.println("Welcome to SimpleClient.");
    new SimpleClient();
}

public void windowActivated(WindowEvent e){}
public void windowClosed(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}
public void windowDeiconified(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowOpened(WindowEvent e){}
public void windowClosing(WindowEvent e)
{
    if (e.getSource()==workingFrame)
    {
        try
        {
            cl.leave(GMAddr);
            cl.shutdown();
        } catch (Exception ex)
        {
            System.out.println("Error while closing: "+ex.getMessage());
        }
    }
    System.out.println("SimpleClient closed.");System.exit(0);
}

public void mouseReleased(MouseEvent e){}
public void mouseEntered(MouseEvent e){}
public void mouseExited(MouseEvent e){}
public void mousePressed(MouseEvent e){}
public void mouseClicked(MouseEvent e){}


private java.net.InetAddress getIP(String host) throws java.net.
    UnknownHostException
{
    return java.net.InetAddress.getByName(host);
}



public void keyTyped(KeyEvent e){}
public void keyPressed(KeyEvent e){}


public void keyReleased(KeyEvent e)
{
    if (e.getKeyCode()==e.VK_ENTER) actionPerformed(new ActionEvent(okButton
        ,1001,"fired programmatically."));
}



public void actionPerformed(ActionEvent e)
{
    Object source=e.getSource();
    if (source==okButton)
    {
        okButton.setEnabled(false);
        int port=0;
        int dsPort=0;
        int min=0;
        int max=0;
        currentFrame=configFrame;
```

```
try
{
    try
    {
        port = Integer.parseInt(portField.getText());
    } catch (NumberFormatException ex)
    {
        throw new RuntimeException("Invalid TCP port number: "+portField.
            getText());
    }
    if (port<10 || port>65535)
        throw new RuntimeException("Invalid TCP port number: "+port);
    try
    {
        dsPort = Integer.parseInt(dsPortField.getText());
    } catch (NumberFormatException ex)
    {
        throw new RuntimeException("Invalid TCP port number: "+
            dsPortField.getText());
    }
    if (dsPort<10 || dsPort>65535)
        throw new RuntimeException("Invalid TCP port number: "+dsPort);

    try
    {
        min = Integer.parseInt(minField.getText());
    } catch (NumberFormatException ex)
    {
        throw new RuntimeException("Invalid Minimum: "+minField.getText()
            );
    }
    if (min<2 || min>10000)
        throw new RuntimeException("Invalid Minumum: "+min);

    try
    {
        max = Integer.parseInt(maxField.getText());
    } catch (NumberFormatException ex)
    {
        throw new RuntimeException("Invalid Maximum: "+maxField.getText()
            );
    }
    if (max<2 || max>10000)
        throw new RuntimeException("Invalid Maxumum: "+max);

    if (max<min)
        throw new RuntimeException("Minimum less than Maximum.");

    configFrame.hide();
    System.out.println("Connecting to Discovery Server...");
    Socket dsSocket=new Socket();
    dsSocket.setKeepAlive(true);
    dsSocket.setTcpNoDelay(true);
    dsSocket.setSoTimeout(10000);
    dsSocket.connect(new InetSocketAddress(getIP(dsField.getText()),
        dsPort),10000);
    ObjectOutputStream oos=new ObjectOutputStream(dsSocket.
        getOutputStream());
    ObjectInputStream ois = new ObjectInputStream(dsSocket.getInputStream
        ());
    System.out.println("Requesting Group of Size "+min+" - "+max+".");
    protocols.Message msg=new protocols.discover.DiscoveryRequest(min,max
        );
    oos.writeObject(msg);
    oos.flush();
    msg = (protocols.Message) ois.readObject();
```

```
            if (!(msg instanceof protocols.discover.DiscoveryReply))
            {
                throw new RuntimeException("Bad response from Discovery Server.")
                    ;
            }
            java.util.Vector GMRecords=((protocols.discover.DiscoveryReply)msg).
                GMRecords;
            dsSocket.close();
            if (GMRecords.size()<1)
            {
                throw new RuntimeException("No suitable groups found. Try running
                    Group Manager.");
            }
            System.out.println("Found "+GMRecords.size()+" suitable Groups.
                Choosing the first ...");
            discoveryserver.GMRecord gmr=(discoveryserver.GMRecord)GMRecords.
                elementAt(0);
            GMAddr=gmr.getIpAndPort();

            System.out.println("Reported group size: "+gmr.getGroupSize()+" at "+
                GMAddr+" ...");
            cl = new client.Client(port);
            makeWorkingFrame();
            workingFrame.pack();
            workingFrame.show();
            currentFrame=workingFrame;
            cl.addChangeListener(this);
            cl.start();
            cl.register(min,max,GMAddr);
        } catch (Exception ex)
        {
            ex.printStackTrace();
            String msg="Error!\n"+ex.getMessage();
            JOptionPane.showMessageDialog(currentFrame, msg, "Error", JOptionPane
                .ERROR_MESSAGE);
            okButton.setEnabled(true);
            return;
        }
    } else if (source==pingButton)
    {
        if (cl.getPingingEnabled())
        {
            pingButton.setText(" Switch Pinging On ");
            pingButton.setSelected(false);
            cl.setPingingEnabled(false);
        } else
        {
            pingButton.setText(" Switch Pinging Off ");
            pingButton.setSelected(true);
            cl.setPingingEnabled(true);
        }
    }
}




private void makeConfigFrame()
{
    configFrame=new JFrame("Credential Obtaining Client");
    configFrame.addWindowListener(this);
    Font bigFont = new Font("monospaced", Font.BOLD, 18);
    Font smallFont = new Font("monospaced", Font.PLAIN, 14);

    // put min/max groupsizes in a Box
```

```
Box  minMaxBox = new Box(BoxLayout.X_AXIS);
JLabel portLabel = new JLabel("TCP port:");
portLabel.setFont(smallFont);
portLabel.setMaximumSize(portLabel.getPreferredSize());
portField=new JTextField();
portField.setFont(smallFont);
portField.setColumns(5);
portField.setMaximumSize(portField.getPreferredSize());
JLabel minLabel = new JLabel("minimum:");
minLabel.setFont(smallFont);
minLabel.setMaximumSize(minLabel.getPreferredSize());
minField=new JTextField();
minField.setFont(smallFont);
minField.setColumns(5);
minField.setMaximumSize(minField.getPreferredSize());
JLabel maxLabel = new JLabel("maximum:");
maxLabel.setFont(smallFont);
maxLabel.setMaximumSize(maxLabel.getPreferredSize());
maxField=new JTextField();
maxField.setFont(smallFont);
maxField.setColumns(5);
maxField.setMaximumSize(maxField.getPreferredSize());
minMaxBox.add(minMaxBox.createHorizontalGlue());
minMaxBox.add(portLabel);
minMaxBox.add(minMaxBox.createHorizontalStrut(10));
minMaxBox.add(portField);
minMaxBox.add(minMaxBox.createHorizontalStrut(20));
minMaxBox.add(minLabel);
minMaxBox.add(minMaxBox.createHorizontalStrut(10));
minMaxBox.add(minField);
minMaxBox.add(minMaxBox.createHorizontalStrut(20));
minMaxBox.add(maxLabel);
minMaxBox.add(minMaxBox.createHorizontalStrut(10));
minMaxBox.add(maxField);
minMaxBox.add(minMaxBox.createHorizontalGlue());

// put minMax box in a Panel
JPanel minMaxPanel=new JPanel();
minMaxPanel.add(minMaxBox);
minMaxPanel.setBorder(BorderFactory.createTitledBorder("Desired Group Size
    Bounds"));


// put discovery server fields in a box
Box   dsBox = new Box(BoxLayout.X_AXIS);
JLabel dsLabel=new JLabel("Address:");
dsLabel.setFont(smallFont);
dsField=new JTextField();
dsField.setFont(smallFont);
dsField.setColumns(30);
dsField.setMaximumSize(dsField.getPreferredSize());
dsField.addKeyListener(this);
JLabel dsPortLabel=new JLabel("dsPort:");
dsPortLabel.setFont(smallFont);
dsPortField=new JTextField("2004");
dsPortField.setFont(smallFont);
dsPortField.setColumns(10);
dsPortField.setMaximumSize(dsPortField.getPreferredSize());
dsPortField.addKeyListener(this);
dsBox.add(dsBox.createHorizontalGlue());
dsBox.add(dsLabel);
dsBox.add(dsBox.createHorizontalStrut(10));
dsBox.add(dsField);
dsBox.add(dsBox.createHorizontalGlue());
dsBox.add(dsBox.createHorizontalStrut(10));
dsBox.add(dsBox.createHorizontalGlue());
```

```
        dsBox.add(dsPortLabel);
        dsBox.add(dsBox.createHorizontalStrut(10));
        dsBox.add(dsPortField);
        dsBox.add(dsBox.createHorizontalGlue());


        // put discovery server box in the dsPanel
        JPanel dsPanel=new JPanel();
        dsPanel.add(dsBox);
        dsPanel.setBorder(BorderFactory.createTitledBorder("Discovery Serves"));


        // put ok button in a box
        Box okBox = new Box(BoxLayout.X_AXIS);
        okButton=new JButton(" OK ");
        okButton.setFont(bigFont);
        okButton.addActionListener(this);
        okButton.setDefaultCapable(false);
        okBox.add(okBox.createHorizontalGlue());
        okBox.add(okButton);
        okBox.add(okBox.createHorizontalStrut(10));

        // put everything in the main box
        Box mainBox = new Box(BoxLayout.Y_AXIS);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(minMaxPanel);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(dsPanel);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(okBox);

        //add the main box to this Frame
        configFrame.getContentPane().add(mainBox);
    }

    private void updateProgressBar()
    {
        progressBar.setValue(clientRecords.size());
        progressBar.setString(progressBar.getValue()+"/"+progressBar.getMaximum()+": 
            waiting for another "+(progressBar.getMaximum()-progressBar.getValue())+" 
            users.");
        if (clientRecords.size()==progressBar.getMaximum()) progressBar.setString("
            Group full. Ready to obtain credential.");
    }

    private void addPeer(groupmanager.ClientRecord cr, String message)
    {
        int index=getClientIndex(cr.ipap);
        if (index!=-1) return;
        Box clientBox = new Box(BoxLayout.X_AXIS);
        JLabel clientLabel=new JLabel(cr.ipap.toString());
        Font smallFont = new Font("monospaced", Font.PLAIN, 14);
        clientLabel.setFont(smallFont);
        JLabel clientComment=new JLabel(message);
        clientComment.setFont(smallFont);
        clientStati.addElement(clientComment);
        clientBox.add(clientBox.createHorizontalGlue());
        clientBox.add(clientLabel);
        clientBox.add(clientBox.createHorizontalStrut(10));
        clientBox.add(clientComment);
        clientBox.add(clientBox.createHorizontalGlue());
        clientBoxes.add(clientBox);
        clientPanelBox.add(clientBox);
        clientPanelBox.add(clientPanelBox.createVerticalStrut(10));
```

```
        clientRecords.addElement(cr);
        updateProgressBar();
    }


    private void removePeer(groupmanager.ClientRecord cr, String message)
    {
        int index=getClientIndex(cr.ipap);
        if (index==-1) return;
        clientPanelBox.remove((JComponent)clientBoxes.elementAt(index));
        clientBoxes.removeElementAt(index);
        clientStati.removeElementAt(index);
        clientRecords.removeElementAt(index);
        updateProgressBar();
    }

    public void stateChanged(javax.swing.event.ChangeEvent e)
    {
        if (e instanceof client.JoinedGroupEvent)
        {
            client.JoinedGroupEvent jge=(client.JoinedGroupEvent)e;
            progressBar.setMaximum(jge.getGroupSize());
            progressBar.setString("Joined Group of Size "+jge.getGroupSize());
        } else if (e instanceof client.UpdateEvent)
        {
            client.UpdateEvent ue=(client.UpdateEvent)e;
            if (ue.present())
            {
                addPeer(ue.getClientRecord(),ue.getMessage());

            } else
            {
                removePeer(ue.getClientRecord(),ue.getMessage());
            }
        } else if (e instanceof client.ClientConnectionChangeEvent)
        {
            client.ClientConnectionChangeEvent ccce=(client.
                ClientConnectionChangeEvent)e;
            IpAndPort ipap=ccce.getClientIpap();
            int index=getClientIndex(ipap);
            if (index==-1) return;
            if (ccce.isFatal())
            {
                ((JLabel)clientStati.elementAt(index)).setForeground(java.awt.Color.
                    RED);
            } else
            {
                ((JLabel)clientStati.elementAt(index)).setForeground(java.awt.Color.
                    BLACK);
            }
            ((JLabel)clientStati.elementAt(index)).setText(ccce.getState());
        }
        workingFrame.pack();
        workingFrame.validate();
    }

    private int getClientIndex(IpAndPort ipap)
    {
        java.util.Enumeration enum=clientRecords.elements();
        int i=0;
        while (enum.hasMoreElements())
        {
            if (((groupmanager.ClientRecord)enum.nextElement()).ipap.toString().
                equals(ipap.toString())) return i;
            i++;
        }
```

```
        return -1;
    }

    private void makeWorkingFrame()
    {
        workingFrame=new JFrame("Group Overview");
        workingFrame.addWindowListener(this);
        Font smallFont = new Font("monospaced", Font.BOLD, 14);
        // put progressbar in a box
        Box  titleBox = new Box(BoxLayout.X_AXIS);
        progressBar=new JProgressBar(0,2);
        progressBar.setStringPainted(true);
        progressBar.setString("Registering...");

        titleBox.add(titleBox.createHorizontalGlue());
        titleBox.add(titleBox.createHorizontalStrut(10));
        titleBox.add(progressBar);
        titleBox.add(titleBox.createHorizontalStrut(10));
        titleBox.add(titleBox.createHorizontalGlue());


        // prepare a vertical box for clients and put it in a panel
        clientPanelBox=new Box(BoxLayout.Y_AXIS);
        clientPanelBox.add(clientPanelBox.createVerticalStrut(10));
        JPanel clientPanel=new JPanel();
        clientPanel.setBorder(BorderFactory.createTitledBorder("Group Members"));
        clientPanel.add(clientPanelBox);

        // put ping button in a box
        Box pingBox = new Box(BoxLayout.X_AXIS);
        pingButton=new JToggleButton(" Switch Pinging On ");
        pingButton.setFont(smallFont);
        pingButton.addActionListener(this);
//          pingButton.setDefaultCapable(false);
        pingBox.add(pingBox.createHorizontalGlue());
        pingBox.add(pingButton);
        pingBox.add(pingBox.createHorizontalStrut(10));



        // put everything in the main box
        Box mainBox = new Box(BoxLayout.Y_AXIS);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(titleBox);
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(clientPanel);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(pingBox);
        mainBox.add(mainBox.createVerticalGlue());

        //add the main box to this Frame
        workingFrame.getContentPane().add(mainBox);
    }


}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
```

```
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

import java.io.*;
import java.util.Vector;
import discoveryserver.DiscoveryServer;

/**
  *   This class is a relatively simple program that starts a {@link DiscoveryServer.
      DiscoveryServer} daemon;
  */

public class SimpleDS
{
    private SimpleDS(int port, PrintWriter log)
    {
        try
        {
            System.out.println("Welcome to SimpleDS.");
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            DiscoveryServer ds=new DiscoveryServer(port, log);
            ds.start();
            System.out.println("Press Enter to stop SimpleDS.");
            br.readLine();
            ds.shutdown();
            System.out.println("Bye!");

        } catch (Exception e)
        {
            e.printStackTrace();
            System.exit(1);
        }
    }


    public static void main(String[] argv)
    {
        int port=0;
        PrintWriter log=new PrintWriter(System.out, true);
        if (argv.length!=1 && argv.length!=2)
        {
            System.out.println("Usage: java SimpleDS portnumber [logfile]");
            System.exit(1);
        }
        try
        {
            port=Integer.parseInt(argv[0]);
            if (port<1 || port>65535)
            {
                System.out.println("Error: "+port+" is not a valid port number.");
                System.exit(1);
            }
        } catch (NumberFormatException e)
        {
            System.out.println("Error: \""+argv[0]+"\" does not appear to be a port
                number.");
```

```
            System.exit(1);
        }
        if (argv.length==2)
        {
            try
            {
                log=new PrintWriter(new FileOutputStream(argv[1], true));

            } catch (IOException e)
            {
                e.printStackTrace();
                System.out.println("\nError opening "+argv[1]);
                System.exit(1);
            }
        }
        new SimpleDS(port, log);
    }
}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.util.Vector;
import java.awt.Font;
import protocols.IpAndPort;

/**
  *   This class is a program that offers a user-friendly interface that controls a {
      @link groupmanager.GroupManager} daemon;
  */

public class SimpleGM implements WindowListener, KeyListener, MouseListener,
    ActionListener, javax.swing.event.ChangeListener
{
    private JFrame configFrame, workingFrame, currentFrame;
    private JTextField tcpPortField, groupSizeField, dsField, portField;
    private JLabel[] dsStatusLabel;
    private JProgressBar progressBar;
    private DefaultListModel dsListModel;
    private JList dsList;
    private JButton okButton;

    private Vector clientRecords, clientStati, clientBoxes; // for working frame (
        clientRecords for housekeeping)
    Box     clientPanelBox;    // for workingFrame


    public SimpleGM()
```

```
{
    clientStati=new Vector();
    clientRecords=new Vector();
    clientBoxes=new Vector();
    configFrame=new JFrame("Group Management Console");
    dsListModel=new DefaultListModel();
    configFrame.addWindowListener(this);
    //adopt to look-and-feel
    try
    {
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e)
    {
        e.printStackTrace();
    }

    makeConfigFrame();
    currentFrame=configFrame;
    configFrame.pack();
    configFrame.show();
}


private void makeConfigFrame()
{
   Font bigFont = new Font("monospaced", Font.BOLD, 18);
   Font smallFont = new Font("monospaced", Font.PLAIN, 14);
    // put tcp port and group size in a panel
    JPanel groupManagementPanel=new JPanel();

    // put TCP port in a box
    Box  tcpPortBox = new Box(BoxLayout.X_AXIS);
    JLabel tcpPortlabel = new JLabel("TCP port:");
    tcpPortlabel.setFont(smallFont);
    tcpPortlabel.setMaximumSize(tcpPortlabel.getPreferredSize());
    tcpPortField=new JTextField();
    tcpPortField.setFont(smallFont);
    tcpPortField.setColumns(5);
    tcpPortField.setMaximumSize(tcpPortField.getPreferredSize());
    tcpPortBox.add(tcpPortBox.createHorizontalGlue());
    tcpPortBox.add(tcpPortlabel);
    tcpPortBox.add(tcpPortBox.createHorizontalStrut(10));
    tcpPortBox.add(tcpPortField);
    tcpPortBox.add(tcpPortBox.createHorizontalGlue());

    // put Group Size in a box
    Box  groupSizeBox = new Box(BoxLayout.X_AXIS);
    JLabel groupSizelabel = new JLabel("Group Size:");
    groupSizelabel.setFont(smallFont);
    groupSizelabel.setMaximumSize(groupSizelabel.getPreferredSize());
    groupSizeField=new JTextField();
    groupSizeField.setFont(smallFont);
    groupSizeField.setColumns(5);
    groupSizeField.setMaximumSize(groupSizeField.getPreferredSize());
    groupSizeBox.add(groupSizeBox.createHorizontalGlue());
    groupSizeBox.add(groupSizelabel);
    groupSizeBox.add(groupSizeBox.createHorizontalStrut(10));
    groupSizeBox.add(groupSizeField);
    groupSizeBox.add(groupSizeBox.createHorizontalGlue());

    groupManagementPanel.add(tcpPortBox);
    groupManagementPanel.add(groupSizeBox);
    groupManagementPanel.setBorder(BorderFactory.createTitledBorder("Group
        Manager Control Panel"));

    // put discovery server fields in a box
```

```
Box    dsBox = new Box(BoxLayout.X_AXIS);
JLabel dsLabel=new JLabel("Address:");
dsLabel.setFont(smallFont);
dsField=new JTextField();
dsField.setFont(smallFont);
dsField.setColumns(30);
dsField.setMaximumSize(dsField.getPreferredSize());
dsField.addKeyListener(this);
JLabel portLabel=new JLabel("port:");
portLabel.setFont(smallFont);
portField=new JTextField("2004");
portField.setFont(smallFont);
portField.setColumns(10);
portField.setMaximumSize(portField.getPreferredSize());
portField.addKeyListener(this);
dsBox.add(dsBox.createHorizontalGlue());
dsBox.add(dsLabel);
dsBox.add(dsBox.createHorizontalStrut(10));
dsBox.add(dsField);
dsBox.add(dsBox.createHorizontalGlue());
dsBox.add(dsBox.createHorizontalStrut(10));
dsBox.add(dsBox.createHorizontalGlue());
dsBox.add(portLabel);
dsBox.add(dsBox.createHorizontalStrut(10));
dsBox.add(portField);
dsBox.add(dsBox.createHorizontalGlue());

// actual list box
Box    listBox = new Box(BoxLayout.X_AXIS);
dsList=new JList(dsListModel);
dsList.setCellRenderer(new DefaultListCellRenderer());
dsList.setSelectionMode(0);
dsList.addKeyListener(this);
dsList.addMouseListener(this);
dsList.setFont(smallFont);
JScrollPane sP = new JScrollPane();
sP.getViewport().setView(dsList);
sP.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
sP.setViewportBorder(BorderFactory.createTitledBorder("to register with"));
listBox.add(listBox.createHorizontalGlue());
listBox.add(sP);
listBox.add(listBox.createHorizontalGlue());

// put discovery server-related boxes in the dsPanel
JPanel dsPanel=new JPanel();
Box dsMainBox = new Box(BoxLayout.Y_AXIS);
dsMainBox.add(dsBox);
dsMainBox.add(dsMainBox.createVerticalGlue());
dsMainBox.add(dsMainBox.createVerticalStrut(20));
dsMainBox.add(dsMainBox.createVerticalGlue());
dsMainBox.add(listBox);
dsMainBox.add(dsMainBox.createVerticalGlue());
dsPanel.add(dsMainBox);
dsPanel.setBorder(BorderFactory.createTitledBorder("Discovery Servers"));

// put ok button in a box
Box okBox = new Box(BoxLayout.X_AXIS);
okButton=new JButton(" OK ");
okButton.setFont(bigFont);
okButton.addActionListener(this);
okButton.setDefaultCapable(false);
okBox.add(okBox.createHorizontalGlue());
okBox.add(okButton);
okBox.add(okBox.createHorizontalStrut(10));

// put everything in the main box
```

```
        Box mainBox = new Box(BoxLayout.Y_AXIS);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(groupManagementPanel);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(dsPanel);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(okBox);

        //add the main box to this Frame
        configFrame.getContentPane().add(mainBox);
    }

    public static void main(String[] argv)
    {
        System.out.println("Welcome to SimpleGroupManager.");
        new SimpleGM();
    }

    public void windowActivated(WindowEvent e){}
    public void windowClosed(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowOpened(WindowEvent e){}
    public void windowClosing(WindowEvent e){System.out.println("SimpleGroupManager
        closed.");System.exit(0);}

    public void mouseReleased(MouseEvent e){}
    public void mouseEntered(MouseEvent e){}
    public void mouseExited(MouseEvent e){}
    public void mousePressed(MouseEvent e){}
    public void mouseClicked(MouseEvent e)
    {
        Object source=e.getSource();
        if (source==dsList)
        {
            if (e.getClickCount()>1)
            {
                int idx=dsList.getSelectedIndex();
                if (!(idx<0)) dsListModel.removeElementAt(idx);

            }
            return;
        }
    }

    private void addDiscoveryServer()
    {
        try
        {
            int port=0;
            try
            {
                port = Integer.parseInt(portField.getText());
            } catch (NumberFormatException e)
            {
                throw new RuntimeException("Invalid port number: "+portField.getText
                    ());
            }
            if (port<10 || port>65535)
                throw new RuntimeException("Invalid port number: "+port);
            if (dsField.getText().length()==0)
                throw new RuntimeException("No Address Given!");
```

```
            IpAndPort addr=new IpAndPort(getIP(dsField.getText()),port);
            for (int i=0; i<dsListModel.size(); i++)
                if ( ((IpAndPort)dsListModel.get(i)).equals(addr))
                    throw new RuntimeException("Discovery Server already exists in
                        the List!");
            dsListModel.addElement(addr);
            dsField.setText("");
        } catch (Exception ex)
        {
            dsField.setText("");
            String msg="Could not add Discovery Server to List!\n"+ex.getMessage();
            JOptionPane.showMessageDialog(currentFrame, msg, "Error", JOptionPane.
                ERROR_MESSAGE);
        }
    }

    private java.net.InetAddress getIP(String host) throws java.net.
        UnknownHostException
    {
        return java.net.InetAddress.getByName(host);
    }


    public void keyTyped(KeyEvent e){}
    public void keyPressed(KeyEvent e){}


    public void keyReleased(KeyEvent e)
    {
        Object source=e.getSource();
        if (source==dsField || source==portField)
        {
            if (e.getKeyCode()==e.VK_ENTER)
            {
                addDiscoveryServer();
            }
            return;
        }

        if (source==dsList)
        {
            if (e.getKeyCode()==e.VK_DELETE)
            {
                int idx=dsList.getSelectedIndex();
                if (!(idx<0)) dsListModel.removeElementAt(idx);
            }
            return;
        }

    }


    public void actionPerformed(ActionEvent e)
    {
        Object source=e.getSource();
        if (source==okButton)
        {
            int port=0;
            int groupSize=0;
            groupmanager.GroupManager gm=null;
            try
            {
                try
                {
                    port = Integer.parseInt(tcpPortField.getText());
```

```
                } catch (NumberFormatException ex)
                {
                    throw new RuntimeException("Invalid TCP port number: "+
                        tcpPortField.getText());
                }
                if (port<10 || port>65535)
                    throw new RuntimeException("Invalid TCP port number: "+port);
                try
                {
                    groupSize = Integer.parseInt(groupSizeField.getText());
                } catch (NumberFormatException ex)
                {
                    throw new RuntimeException("Invalid Group Size: "+groupSizeField.
                        getText());
                }
                if (groupSize<2 || groupSize>10000)
                    throw new RuntimeException("Invalid Group Size: "+groupSize);


                if (dsListModel.getSize()<1)
                    throw new RuntimeException("No Discovery Server Given!");
                groupmanager.Init init=new groupmanager.Init(port,groupSize);
                for (java.util.Enumeration addresses=dsListModel.elements();addresses
                    .hasMoreElements();init.addDS((IpAndPort)addresses.nextElement()))
                    ;
                System.out.println("Starting Group Manager Daemon...");
                gm = new groupmanager.GroupManager(init);
            } catch (Exception ex)
            {
//                ex.printStackTrace();
                String msg="Error!\n"+ex.getMessage();
                JOptionPane.showMessageDialog(currentFrame, msg, "Error", JOptionPane
                    .ERROR_MESSAGE);
                return;
            }

            configFrame.hide();
            makeWorkingFrame(gm);
            workingFrame.show();
            workingFrame.pack();


            gm.addChangeListener(this);
            gm.start();
        }
    }


    private void makeWorkingFrame(groupmanager.GroupManager gm)
    {

            workingFrame=new JFrame("Group Manager");
            Font smallFont = new Font("monospaced", Font.PLAIN, 14);
            Font bigFont = new Font("monospaced", Font.BOLD, 18);
            // put title in a box
            Box  titleBox = new Box(BoxLayout.X_AXIS);
            progressBar=new JProgressBar(0,gm.getGroupSize());
            progressBar.setStringPainted(true);
            progressBar.setString("Waiting for users to join.");


            titleBox.add(titleBox.createHorizontalGlue());
            titleBox.add(progressBar);
            titleBox.add(titleBox.createHorizontalGlue());

            // put discovery server list in a panel
```

```
        JPanel dsPanel=new JPanel();
        Box    dsPanelBox=new Box(BoxLayout.Y_AXIS);

        // make an array of boxes, one for each DS
        dsStatusLabel=new JLabel[dsListModel.getSize()];

        for (int i=0; i<dsListModel.getSize(); i++)
        {
            Box dsBox = new Box(BoxLayout.X_AXIS);
            JLabel dsLabel = new JLabel(dsListModel.elementAt(i).toString());
            dsLabel.setFont(smallFont);
            dsLabel.setMaximumSize(dsLabel.getPreferredSize());
            dsStatusLabel[i]=new JLabel("Initializing...");
            dsStatusLabel[i].setFont(smallFont);
            dsStatusLabel[i].setForeground(java.awt.Color.BLACK);
            dsStatusLabel[i].setMaximumSize(dsStatusLabel[i].getPreferredSize());
            dsBox.add(dsBox.createHorizontalStrut(20));
            dsBox.add(dsLabel);
            dsBox.add(dsBox.createHorizontalStrut(10));
            dsBox.add(dsStatusLabel[i]);
            dsBox.add(dsBox.createHorizontalGlue());
            dsPanelBox.add(dsPanelBox.createVerticalStrut(10));
            dsPanelBox.add(dsBox);
        }
        dsPanel.add(dsPanelBox);
        dsPanel.setBorder(BorderFactory.createTitledBorder("Discovery Server(s)")
            );


        // prepare a vertical box for clients and put it in a panel
        clientPanelBox=new Box(BoxLayout.Y_AXIS);
        clientPanelBox.add(clientPanelBox.createVerticalStrut(10));
        JPanel clientPanel=new JPanel();
        clientPanel.setBorder(BorderFactory.createTitledBorder("Group Members"));
        clientPanel.add(clientPanelBox);



        // put everything in the main box
        Box mainBox = new Box(BoxLayout.Y_AXIS);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(titleBox);
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(dsPanel);
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(mainBox.createVerticalStrut(10));
        mainBox.add(mainBox.createVerticalGlue());
        mainBox.add(clientPanel);
        mainBox.add(mainBox.createVerticalGlue());

        workingFrame.getContentPane().add(mainBox);

        //add the main box to this Frame
        workingFrame.getContentPane().add(mainBox);
}



private void addClient(groupmanager.ClientRecord cr, String message)
{
    int index=getClientIndex(cr.ipap);
```

```
    if (index!=-1) return;
    clientRecords.addElement(cr);
    Box clientBox = new Box(BoxLayout.X_AXIS);
    JLabel clientLabel=new JLabel(cr.ipap.toString());
    Font smallFont = new Font("monospaced", Font.PLAIN, 14);
    clientLabel.setFont(smallFont);
    JLabel clientComment=new JLabel(message);
    clientComment.setFont(smallFont);
    clientStati.addElement(clientComment);
    clientBox.add(clientBox.createHorizontalGlue());
    clientBox.add(clientLabel);
    clientBox.add(clientBox.createHorizontalStrut(10));
    clientBox.add(clientComment);
    clientBox.add(clientBox.createHorizontalGlue());
    clientBoxes.add(clientBox);
    clientPanelBox.add(clientBox);
    clientPanelBox.add(clientPanelBox.createVerticalStrut(10));
    updateProgressBar();
}

private void removeClient(groupmanager.ClientRecord cr, String message)
{
    int index=getClientIndex(cr.ipap);
    if (index==-1) return;
    clientPanelBox.remove((JComponent)clientBoxes.elementAt(index));
    clientBoxes.removeElementAt(index);
    clientStati.removeElementAt(index);
    clientRecords.removeElementAt(index);
    updateProgressBar();
}


public void stateChanged(javax.swing.event.ChangeEvent e)
{
    if (e instanceof groupmanager.DSEvent)
    {
        groupmanager.DSEvent dse=(groupmanager.DSEvent)e;
        dsStatusLabel[dse.getIndex()].setText(dse.getState());
        if (dse.getFatal())
        {
            dsStatusLabel[dse.getIndex()].setForeground(java.awt.Color.RED);
        }
        workingFrame.pack();
        workingFrame.validate();
    } else if (e instanceof groupmanager.UpdateEvent)
    {
        groupmanager.UpdateEvent ue=(groupmanager.UpdateEvent)e;
        if (ue.present())
        {
            addClient(ue.getClientRecord(), ue.getMessage());
        } else
        {
            removeClient(ue.getClientRecord(), ue.getMessage());
        }
    } else if (e instanceof groupmanager.ClientConnectionChangeEvent)
    {
        groupmanager.ClientConnectionChangeEvent ccce=(groupmanager.
            ClientConnectionChangeEvent)e;
        IpAndPort ipap=ccce.getClientIpap();
        int index=getClientIndex(ipap);
        if (index==-1) return;
        if (ccce.isFatal())
        {
            ((JLabel)clientStati.elementAt(index)).setForeground(java.awt.Color.
                RED);
        }
```

```
            ((JLabel)clientStati.elementAt(index)).setText(ccce.getState());
        }
    }

    private void updateProgressBar()
    {
        progressBar.setValue(clientRecords.size());
        progressBar.setString(progressBar.getValue()+"/"+progressBar.getMaximum()+": 
            waiting for another "+(progressBar.getMaximum()-progressBar.getValue())+" 
            users.");
        if (clientRecords.size()==progressBar.getMaximum()) progressBar.setString("
            Group full. Initiating procedure.");
    }

    private int getClientIndex(IpAndPort ipap)
    {
        java.util.Enumeration enum=clientRecords.elements();
        int i=0;
        while (enum.hasMoreElements())
        {
            if (((groupmanager.ClientRecord)enum.nextElement()).ipap.toString().
                equals(ipap.toString())) return i;
            i++;
        }
        return -1;
    }



}

/*
    Copyright (C) 2003 Andreas Pashalidis

    This library is free software; you can redistribute it and/or
    modify it under the terms of the GNU Lesser General Public
    License as published by the Free Software Foundation; either
    version 2.1 of the License, or (at your option) any later version.

    This library is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public
    License along with this library; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

import java.io.*;
import java.util.Vector;
import issuer.Issuer;

/**
  *   This class is a relatively simple program that starts a {@link DiscoveryServer.
      DiscoveryServer} daemon;
  */

public class SimpleIssuer
{
    private SimpleIssuer(int port, PrintWriter log)
    {
        try
        {
            System.out.println("Welcome to SimpleIssuer.");
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
```

```
            Issuer is=new Issuer(port, log);
            is.start();
            System.out.println("Press␣Enter␣to␣stop␣SimpleIssuer.");
            br.readLine();
            is.shutdown();
            System.out.println("Bye!");

        } catch (Exception e)
        {
            e.printStackTrace();
            System.exit(1);
        }
    }


    public static void main(String[] argv)
    {
        int port=0;
        PrintWriter log=new PrintWriter(System.out, true);
        if (argv.length!=1 && argv.length!=2)
        {
            System.out.println("Usage:␣java␣SimpleIssuer␣portnumber␣[logfile]");
            System.exit(1);
        }
        try
        {
            port=Integer.parseInt(argv[0]);
            if (port<1 || port>65535)
            {
                System.out.println("Error:␣"+port+"␣is␣not␣a␣valid␣port␣number.");
                System.exit(1);
            }
        } catch (NumberFormatException e)
        {
            System.out.println("Error:␣\""+argv[0]+"\"␣does␣not␣appear␣to␣be␣a␣port␣
                number.");
            System.exit(1);
        }
        if (argv.length==2)
        {
            try
            {
                log=new PrintWriter(new FileOutputStream(argv[1], true));

            } catch (IOException e)
            {
                e.printStackTrace();
                System.out.println("\nError␣opening␣"+argv[1]);
                System.exit(1);
            }
        }
        new SimpleIssuer(port, log);
    }
}
```