# Authorization Management Framework

by

Pornthep Narula

A Thesis
Submitted to the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements
For the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, CANADA

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

# Authorization Management Framework

BY

Pornthep Narula

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

of

MASTER OF SCIENCE

# Abstract

Information security requirements cover origin authentication, privacy/confidentiality, integrity, and availability. A complete security process is a risk management process, which spans prevention, protection, detection, response, and recovery. Security mechanisms are usually split into authentication (i.e. are you speaking the truth?), authorization (i.e. are you allowed to do something?), and accounting (i.e. who did what, where, when, and how?). Authentication and authorization together partially address prevention and protection, while accounting partially addresses detection and response.

Authentication and authorization protocols of interest in this work are those based on public-key digital signatures. The authorization management problem under focus here deals with sessional and transactional authorization plus some parts of accounting.

It has been shown in this thesis that public-key cryptography can be practically and effectively used as a tool in an electronic authorization management solution that is geared towards balancing the needs for privacy protection and accountability preservation. Also, it has been shown how such a solution may be deployed in a manner that could reduce the number of credentials and/or secrets a user has to carry and/or remember in order to utilize the authorization management system.

In addition, a reference implementation of the proposed framework is provided along with a baseline performance benchmarking results. Interesting areas for potential future works such as long-lived signatures, attribute certification, encryption key-pairs management and biometrics authentication are also discussed.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Information security requirements cover origin authentication, privacy/confidentiality, integrity, and availability. A complete security process is a risk management process, which spans prevention, protection, detection, response, and recovery. Security mechanisms are usually split into authentication (i.e. are you speaking the truth?), authorization (i.e. are you allowed to do something?), and accounting (i.e. who did what, where, when, and how?). Authentication and authorization together partially address prevention and protection, while accounting partially addresses detection and response. A system that implements these mechanisms is often called *"access-control system"*.

For detailed information on security and cryptography, please refer to [1, 2]. A brief review on public key infrastructure technologies is provided in Chapter 2.

Authentication and authorization protocols of interest in this work are those based on public-key digital signatures. However, there will be brief discussions in Chapter 6 on other authentication methods and public-key encryption.

## 1.1 Problem Statement

The authorization management problem under focus here deals with sessional and transactional authorization plus some parts of accounting. Specifically, I am trying to answer the following questions:

1. *Can public-key cryptography be practically and effectively used as a tool in an electronic authorization management solution that is geared towards balancing the needs for privacy and accountability? If so then, how?*

2. *Using such an authorization management solution, is it technically feasible to reduce the number of physical credentials one has to carry (e.g. cards and keys) and/or secrets to remember (e.g. user-names and pass-codes) in order to perform activities and/or access resources/services that require authentication and authorization? If so then, how?*

## 1.2 Design Principles

**Privacy and Accountability Balance:** Rather than falling into the extremes and advocates one over the other, the proposed solution must strive to achieve a balance between protection of privacy (be it personal privacy, business privacy, or political privacy [1]) and accountability.

**Human In-control:** *Well-informed* individuals are capable of making, and being responsible for, decisions in matters that affect their rights and responsibilities. Since any authorization chain—not to mention the design and implementation of related systems—can always be traced back to an individual or a group thereof, human beings should ultimately take control and responsibilities. Thus, I quote Stabell-Kulø *et al.*'s *Open-End Argument for Private Computing*:

"The system should be designed in such a way that in all situations where qualitative assessment of information is needed to make a decision, the user is consulted." [3]

**Simple:** Some of the most frequent offenders in security breaches are the users (and sometimes even the system administrators themselves). This has largely been due to the lack of proper understanding and appreciation in the enforced security policy, process, and mechanisms. Such misunderstanding usually lead to confusion and frustration, which eventually resulted in the user accidentally/intentionally circumventing the security mechanisms. So, the simpler a security system is, the easier (read *less costly*) it is to make users understand and comply with the policy. Thus, I quote Albert Einstein:

"Everything should be made as simple as possible, but no simpler!"

## 1.3 Terminology Usage

For the sake of clarity, applications of public-key cryptography will be divided into two types: digital signature and data encryption. Digital signature applications are those that uses public-key cryptography for origin authentication and integrity. Data encryption applications are those that uses public-key cryptography for protection of privacy and/or confidentiality.

For the most part, traditional definitions of terms as provided in standard English dictionaries will be used. For convenient reference, selected dictionary definitions of security-related terms used in this thesis are included in Appendix A. However, there will be terms for which the relationship between their traditional definitions and security-specific definitions may not be so obvious. Some of those terms are defined here as follows [4]:

- An **entity** refers to a person, an organization, or an artifact that is capable of taking a relevant action (e.g. a hardware-server, a software-server, a hardware-client, a software-client, a software agent).
- An **identity** is a defined and specific instance of a specific entity at a particular point in time.
- A **digital persona** is a group of data items that together form a simplified representation of an identity.
- An **identifier** is a data-item or a group of data-items which reliably distinguish the identity of an entity.
- A **role** is a particular presentation of an entity. The mapping of role to entity is many-to-many.
- A **nym** is an identifier of a role (e.g. manager, teacher, student).

In addition, Cryptographic keys and key-pairs used for digital signatures are called **signature keys** and **signature key-pairs**, respectively. On the other hand, keys and key-pairs used for data encryption are called **encryption keys** and **encryption key-pairs**, respectively.

For clarity, other terms will be defined at-first-use throughout the document.

## 1.4 Organization

Chapter 2 provides an introduction to public-key cryptogrpahy and a review on existing standards and their short-comings. To formalize the problem statement, a real-world model of the authorization management problem is presented in Chapter 3. The Authorization Management Framework (AMF) is then proposed in Chapter 4 as a solution, followed by the description of the reference implementation and performance benchmarking results in Chapter 5. The conclusions and potential future works are then discussed in Chapter 6.

# Chapter 2

# Background

This chapter provides a review on the current state-of-the-art in public key cryptography and public key infrastructure (PKI). First, a brief introduction to public key cryptography concept is given. A review of modern PKI-related research done by industry, academia and standards bodies is then presented, followed by the conclusions.

## 2.1 Public Key Cryptography and Public Key Infrastructure (PKI)

*Secret key cryptography* (a.k.a. symmetric key cryptography) employs the same key for both encryption and decryption(see Figure 2.1a). So, each pair (or group) of entities that need to communicate securely would have to share a secret encryption key among themselves. This introduces a number of scalability problems similar to other shared secrets security mechanisms (e.g. password authentication). For example, the number of keys (read *shared secrets*) that each entity has to manage is directly proportional to the number of (group of) entities that it needs to communicate securely with. Add to that the surrounding logistics, like key exchange, key replacement and

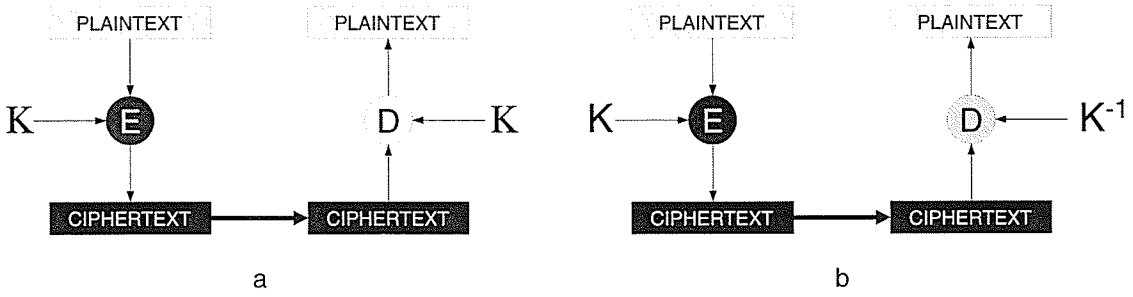


Figure 2.1: Secret key cryptography (a) vs. public key cryptography(b)

key archival and it suffices to say that secret key cryptography by itself does not scale well beyond a small and closed community.

*Public key cryptography* (a.k.a. asymmetric key cryptography), on the other hand, uses two mathematically-related keys, one for encryption and the other for decryption (see Figure 2.1b). One key is to be kept as a secret known only to the key owner, hence the name *private key.* The other key is to be openly distributed to other entities, hence the name *public key.* A message (called *plaintext*) that is encrypted using the private key can only be decrypted by the corresponding public key, thus offering *origin authentication* and *data integrity.* A message encrypted using the public key can only be decrypted by the corresponding private key, thus offering *data confidentiality.*

The principle claim-of-superiority of public key cryptography over secret key cryptography is the elimination of shared secrets. That is, the only secret one needs to keep is the private key, which is never shared with anyone else. So, in theory, one should be able to start communicating securely with anyone else by just exchanging their public keys. In practice, however, it is not that simple.

First, there is the matter of public key cryptographic operations being much slower than secret key cryptographic operations with equivalent strength. This has resulted in work-around procedure that try to minimize the size of data that needs to be fed through the public key encryption/decryption process. For digital signatures, this

involves feeding the plaintext through a one-way collision-free hash function such as SHA-1 before encrypting the output with the sender's private key. The output of this process, called *digital signature*, is then attached to the plaintext and sent to the recipient. The recipient would decrypt the signature with the sender's public key and compare the result with the hash of the received plaintext (see Figure 2.2a).

For confidentiality, this includes encrypting the plaintext using a secret key cryptographic algorithm and encrypting the secret key with the recipient's public key. The two ciphertexts are then sent together to the recipient. The recipient would decrypt the secret key with its private key and use the result to decrypt the entire plaintext (see Figure 2.2b). Examples of such approaches include the Pretty Good Privacy (PGP) system and the S/MIME specification. For on-line and interactive communication protocols (e.g. TLS), this includes using public key cryptography for authentication and exchange of secret session key, which is then used for encrypting and decrypting the actual communication. So, it can be seen that practical uses of public key cryptography have so far been more of a complement to secret key cryptography, rather than a replacement.

Then, there is the matter of being able to reliably obtain the public key of the entities one wants to communicate with, especially in the absence of direct contact between the two communicants. The original proposal by Diffie and Hellman in 1976 was to publish a public directory of public keys along with key owner's names and addresses, much like a telephone book [5]. The publisher of such directory, called the *Public File*, was considered a *Trusted Third Party* (TTP) by Kohnfelder's MIT bachelor's thesis in 1978 [6]. Kohnfelder also proposed that, in order to prevent performance problem for on-line directory access, the publisher digitally pre-signs each record. Kohnfelder then termed the signed data structure a *certificate*. By signing the certificates, the publisher, or certificate issuer, asserts its beliefs in the

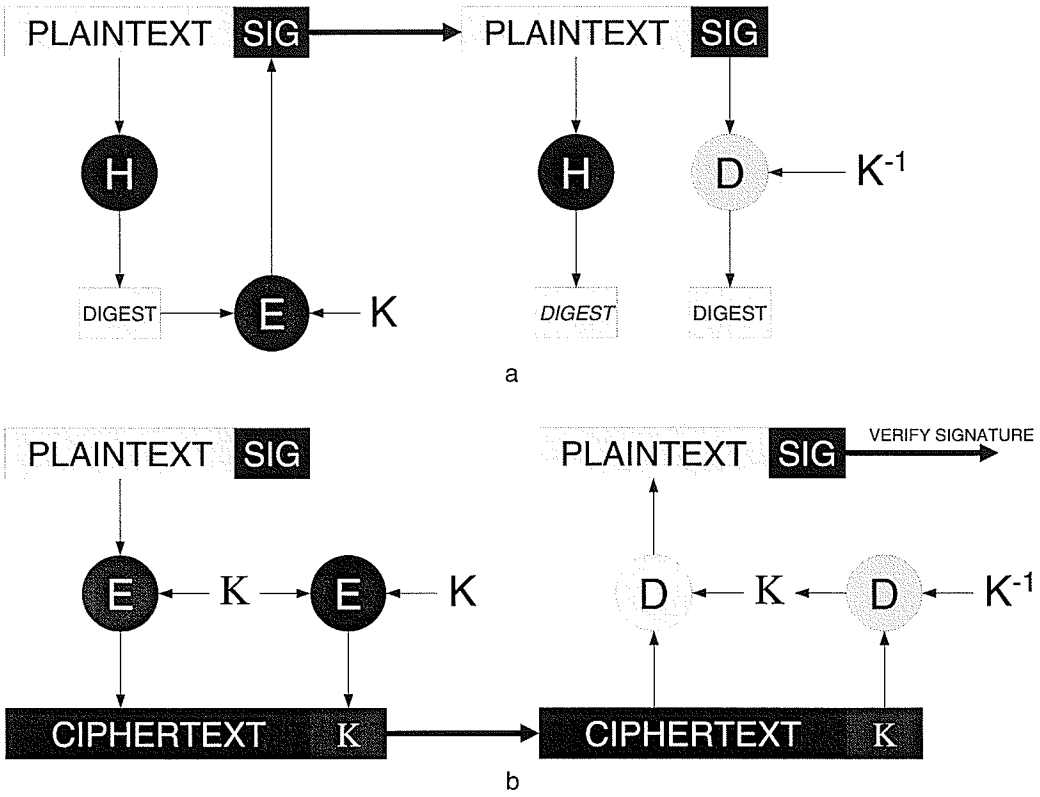


Figure 2.2: Practical public key digital signature (a) and encryption (b)

accuracy of the information contained in those certificates. Relying entities need only to securely obtain the issuer's public key through an off-line mechanism and use it to verify any certificate signed by that certificate issuer. The term *Certification Authority* (CA) was later introduced to call certificate issuer. Certificates, the CA and other supporting human and facilities required for the deployment of public key cryptographic services are collectively called *Public Key Infrastructure* or PKI.

For further information on public key cryptography history and concepts, please refer to [2].

To date, there have been several attempts to develop standards that enable the use of public key certificates (PKCs) to provide data confidentiality, data integrity, origin authentication and non-repudiation services. However, the most widely *"accepted"* suite of standards appears to be the X.509v3 recommendation, jointly developed by the ISO, IEC, ITU and the ANSI-X9 [7]. Selected samples of X.509-base systems and standards are reviewed in Section 2.2, 2.3, 2.4 and 2.5. This is then followed by an introduction to two alternative PKI models in section 2.5 and 2.6 and the concluding remarks.

## 2.2 AT&T's Short-lived Certificates

AT&T's researchers suggested that traditional PKIs are complex and costly due to the overhead and risk involved in dealing with certificates revocation, recovery of compromised CA keys and user mobility [8]. They also asserted that the primary cause is use of certificates with long life-times. They then proposed that the security framework could be simplified through the use of *short-lived certificates*, whose life-times are strongly correlated with the expected length of an entity's job function. They also argued for the use of large pool of key-pairs pre-generated on the server side and randomly picked to create new certificates for users. A user first establishes

its identity with the *Registration Authority* (RA), which would issue a shared secret to the applicant. The user can then uses the shared secret to apply for a new certificate from the *Certificate Authority* (CA) server, which would consult the *Registration Directory* server to verify the client's authenticity, before issuing the certificate.

The advantages of such framework over traditional PKI include elimination of certificate revocation, minimization of user's responsibility over their key-pairs, reduction of key-length requirement and easy recovery from compromised CA keys. Also, by embedding a client's privileges in the certificates, access control systems becomes more efficient, easier to manage and scalable.

The framework was implemented and tested in the Strategic Saltmine Initiative (SSI) project and deployed in AT&T's long-distance network maintenance support systems. The CA provided a web-based interface for client certification service, using a common gateway interface (CGI) program. Each application would publish its user directory to the CA user directory, bringing in their registered users for central management. In their implementation, however, key-pairs were generated by the client's web browser instead of being drawn from a pre-generated key-pool.

The obvious weakness of the short-lived certificate framework as proposed by AT&T is that it cannot scale up to the Internet scale. This is due to the lack of control over both clients and servers. The use of shared secret in the registration and certification process also poses a risk if deployed in an untrusted facility and/or network. The risk and cost involving in maintaining a large pool of pre-generated key-pairs on a server is rather hard to justify. Furthermore, scalability and cost-effectiveness depends mostly on the Registration Directory implementation.

## 2.3 UC Berkeley's CRISIS

As part of the WebOS project at the University of California Berkeley, a wide area security architecture [9] was developed to provide a general, coherent and scalable authentication and access control system for deploying distributed applications on the Internet. The driving force behind the effort was that the administrative overhead of creating and maintaining separate accounts in all independent domains in order to maintain fine-grained access control over remote resources are too prohibitive. Design principles used include redundancy, caching, least privilege, complete accountability, local autonomy and simplicity.

The framework uses X.509-based *identity certificates*, binding entities to public keys, for authentication. Separated *transfer certificates* are used for privilege delegation and accounting. The third type of certificates, called *time certificates*, issued by redundant trusted time-servers, were used for accountability.

Long-lived X.509-based identity certificates are issued by trusted third party (CA) and counter-signed by a locally trusted on-line agent (OLA) with a shorter validity period. Multiple CAs can also be used to create a hierarchical network of trust among autonomous domains.

CRISIS introduces transfer certificates to allow delegation of a signing principal's privileges to a target principal. All programs execute in the context of a security domain. A principal delegates a subset of her privileges to a process using a role by generating an identity certificate and a transfer certificate, both endorsed by a chosen counter-signing on-line agent (OLA). The transfer certificate describes a subset of the principal's rights that are transferred to that role. Authorization is performed using UNIX based access control lists (ACL) for file access and process execution, with each service running its own reference monitor. Both single and multiple inheritance models of role hierarchy are supported for privilege delegation.

A *resource manager* on each machine authenticates the remote principal's identity and determines if the proper access rights are held. The client is responsible for supplying the proper certificate chain to the server during the authentication and authorization process.

WebOS applications are executed in a restricted JVM called Janus and use CRISIS-aware Secure Socket Layer (SSL) for transport confidentiality and integrity. CRISIS was also used in the implementation of the WebFS global file system, also part of the WebOS project, which allows read/write access to files stored across the wide area network.

A major advantage of the framework lies in the fact that splitting CA/OLA makes it harder to subvert the systems while simplifying revocation and reducing the need for synchronous inter-domain communication. The push model also simplifies authorization process.

However, the Network of Workstations (NOW) project, WebOS' parent project, was wrapped up in 1998, thus ending the WebOS and any further research on CRISIS as well. There remain issues that need further refinement, or perhaps even rethinking, within the framework. This includes a flaw in its current implementation that all authentication processes require communication back to the security manager process at the user's home domain to obtain the user's certificates, decreasing availability and mobility.

## 2.4 Globus' GSI

The Globus project focuses on research into computational grids. Computational grids are large-scale distributed computing environments that provide simultaneous and dynamic accesses to large numbers of resources from multiple administrative domains. As such, a computational grid needs a lightweight and flexible authentication

and authorization infrastructure that allows for sharing of resources across organizational boundaries, and yet being able to maintain autonomy over resources according to local policies. Other requirements include a single sign-on capability; protection of credentials (e.g. password, private key, etc.); inter-operability with local security solutions (e.g. KERBEROS, SSH, SSL, etc); exportability (outside the U.S.); open standard conformance (e.g. X.509v3 certificates); support for secure group communication (e.g. among coordinating processes); and support for multiple implementations (e.g. public key and secret key encryption) [10].

Implementing the Grid Security Architecture, the *Grid Security Infrastructure* (GSI) [11] provides common applications (ftp, ssh) and programming toolkits (GSS-API, SSLv3) for constructing secure grid applications. GSI uses X.509-based identity certificate, issued by a Certification Authority (CA), binding entity's globally unique name (identity) to a public key. The SSLv3 protocol's authentication algorithm is used for identity checking. Privilege delegation is achieved by having an entity create and sign a temporary proxy certificate authorizing another entity (e.g. application process) to act on its behalf (or to delegate further).

A local site maps the global identity in the certificate to a local subject name and uses their existing security system (e.g. KERBEROS, DCE and plain text) for authorization. The Generic Authorization and Access Control (GAA) API is provided for certificates chain verification to support multiple CAs. GSI also allows storage of the user's private key both encrypted in local file system and in cryptographic-capable smart card.

As part of the Globus toolkit, GSI has been deployed in the two Partnerships for Advanced Computational Infrastructure (PACI), which are developing grids of supercomputers and storage systems that span over 50 universities and government laboratories.

A major design difference of GSI from CRISIS is the fact that GSI only provides inter-domain authentication solutions and stays clear of intra-site security policy and implementation completely. CRISIS, on the other hand, allows for local policy autonomy but provides a complete (read intrusive) infrastructure for authentication, authorization, accounting and execution mechanisms for all participating sites. Although both extreme approaches clearly have distinct advantages, it would be more desirable to have an infrastructure that provides flexibility and comprehensive support for both inter-domain and intra-domain security.

## 2.5 IETF's PKIX Working Group

It has been observed that the original definition of X.509 certificates ("which key holder had permission to modify which X.500 directory nodes" [12]) and how it is actually being used today ("the key speaks for the named person" [12]) are totally different. It has also been noted that the X.509v3 certificate and CRL profiles are too broad to be directly used to develop independent implementations that will interoperate with on another. This has resulted in the creation of the Public Key Infrastructure (X.509) working group, a.k.a. PKIX, under the Internet Engineering Task Force (IETF). The charter of the PKIX working group was defined as quoted below [13]:

> "The PKIX Working Group was established in the Fall of 1995 with the intent of developing Internet standards needed to support an X.509-based PKI. Several informational and standards track documents in support of the original goals of the WG have been approved by the IESG. The first of these standards, RFC 2459, profiles the X.509 version 3 certificates and version 2 CRLs for use in the Internet. The Certificate Management Pro-

tocol (CMP) (RFC 2510), the Online Certificate Status Protocol (OCSP) (RFC 2560) and the Certificate Management Request Format (CRMF) (RFC 2511) have been approved, as have profiles for the use of LDAP v2 for certificate and CRL storage (RFC 2587) and the use of FTP and HTTP for transport of PKI operations (RFC 2585). RFC 2527, an informational RFC on guidelines for certificate policies and practices also has been published, and the IESG has approved publication of an information RFC on use of KEA (RFC 2528) and is expected to do the same for ECDSA. Work continues on a second certificate management protocol, CMC, closely aligned with the PKCS publications and with the cryptographic message syntax (CMS) developed for S/MIME. A road-map, providing a guide to the growing set of PKIX document, is also being developed as an informational RFC. The working group is now embarking on additional standards work to develop protocols that are either integral to PKI management, or that are otherwise closely related to PKI use. Work is ongoing on alternative certificate revocation methods. There also is work defining conventions for certificate name forms and extension usage for "qualified certificates," certificates designed for use in (legally binding) non-repudiation contexts. Finally, work is underway on protocols for time stamping and data certification. These protocols are designed primarily to support non-repudiation, making use of certificates and CRLs, and are so tightly bound to PKI use that they warrant coverage under this working group.

Additional work be initiated on a profile for X.509 attribute certificates, resulting in a new RFC and, perhaps, in extensions to existing certificate management standards to accommodate differences between

attribute certificates and public key certificates."

The PKIX theory [14], based primarily on X.509v3, categorizes certificate-using systems into Public Key Infrastructure (PKI) and Privilege Management Infrastructure (PMI). PKI is defined as "The set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke PKCs based on public key cryptography." Public Key Certificates (PKCs) are "data structures that bind public key values to subjects," digitally signed by a *trusted* CA with limited validity periods. PKIX's definition of PKI is essentially equivalent to the traditional PKI described earlier. This consists of certification authority, (optional) registration authority, certificate holders, relying parties and repositories (or directories) for certificates and CRLs.

The main specifications document [15] provides a profile for PKCs and CRLs by describing the content format and semantics, as well as degrees of criticality and value constraints for many extensions. In addition, the document also defines Object Identifiers (OID) for a number of well-known encryption algorithms to promote interoperability.

The default name format for identifying the certificate Issuer and Subject is the Distinguished Name (DN). DN is defined as the X.501 type Name, which consists of a sequence of <type,value> attribute pairs. PKIX requires that conforming implementations must support at least the following types: country (C), organization (O), organizational-unit (OU), distinguished name qualifier (DQ), state/province name (L), common name (CN) and serial number (SN). An example of a DN is "C=CA, O=TRLabs, OU=Data Networking, L=Manitoba, CN=Pornthep Narula". The preferred string encoding for DN is UTF8String.

However, PKIX also allows for alternative name formats tailored for Internet applications, by defining extensions called SubjectAltName and IssuerAltName, e.g.

email address (rfc822Name), domain name (dNSName), WWW URL (uniformResourceIdentifier), IP address (iPAddress), etc. This allows for issuing of certificates that *binds* public keys to email addresses for S/MIME, web servers for SSL/TLS and IP addresses for IPSec.

The scope of name in PKIX is local to a particular domain. This means that all names need to be defined and unique only within their own domain. The fact that there are two or more CAs of the same name, each potentially defining unique subject names that overlap with other CAs, in different domains is considered irrelevant. Of course, this implies that those CAs won't be interacting with one another across their domains, e.g. through cross-certification. If inter-domain communication is needed but the uniqueness of the CA's Distinguished Names cannot be guaranteed, a technically possible solution (but not supported by the PKIX standard) is having each CA register for their own unique OID with the ISO, which then could be used in the registeredID type in the IssuerAltName extension of its certificate to uniquely identify themselves with other CAs. But, again, this brings back the matter of accepting the ISO as the single global naming authority.

Another document [16] was under development to profile a type of certificate, termed *Qualified Certificates* (QC) by the European Commission, that could be used to identify a person in supporting non-repudiation of digital signatures in a legal sense. Recommendations are provided for the content of the Issuer and Subject fields, as well as Subject Directory, Certificate Policies and Key Usage extensions. The document also introduces two new private extensions, one for storage of biometric data and the other for storage of statements related to QCs. The biometricInfo extension is defined to store an output of a one-way hash, e.g. SHA-1, of a biometric template suitable for human verification, e.g. portrait photography, handwritten-signature.

A notable point of the document is the disclaimer that appears in the last paragraph of the first section, which goes as follows:

> "That is, this document defines what information should go into Qualified Certificates, but not what that information means. A system that uses Qualified Certificates must define its own semantics for the information in Qualified Certificates. It is expected that laws and corporate policies will make these definitions." [16]

At first glance, such disclaimer seems to make sense for a document whose purpose is to describe the syntax required for technical inter-operability. However, one should be careful in noting that the document is being developed in response to the requirements set by a particular community, i.e. Europe. This means that, even with the disclaimer taken into consideration, the profile may or may not satisfy the requirements of communities with different legal and social frameworks. An alternative profile called Permanent Identifier, as defined in [17], may be more suitable for communities that already have infrastructure for persistently-unique identifier in place, e.g. Canada's SIN number or Thailand's citizenship ID number.

*Trust models* (models used in deciding where a particular entity's trusted CA is located) recognized by PKIX in [14] include hierarchical (e.g. X.500-style, military), local/federation (local CAs cross-certify each other to create certificate paths; a.k.a peer-to-peer model), root repository (e.g. embedding CA certificates in web browsers), and relying party's perspective (the relying party can autonomously choose to trust a particular PKC or domain of PKCs).

*Privilege Management Infrastructure* (PMI) is defined as "The set of hardware, software, people, policies and procedures needed to create, manage, store, distribute and revoke ACs." Attribute Certificates (ACs) are "data structures containing a set of attributes for an end-entity and some other information," [14] digitally signed by the

*Attribute Authority* (AA). The main intentional use of PMI for PKIX is in supporting role-based and rule-based access control. Exclusion of authorization information from PKCs and putting it in separate ACs makes it possible to alter a subject's privileges at any time without affecting the life-time of the PKC and the key-pair certified by it. Another rationale behind this separation is the fact that "the PKC issuer is not usually authoritative for the authorization information."

Another document [18] describes the format and semantics for ACs. It provides standard attribute types for inter-operability (e.g. group membership, role allocation, clearance level, etc.) and also allows for private attributes definition and defines extensions that allow AA to, in addition to long-live ACs, issue short-lived ACs that need no revocation infrastructure, to target an AC at specified servers, and to support pull-model for certificate distribution. Naming format in AC version 1 include referencing to a PKC's certificate ID (the preferred format within PKIX), or a subject's name (i.e. Distinguished Name or AltName). AC version 2 (the use of which is currently discouraged within PKIX) also allows for the use of the hash value of the subject's public key or entire PKC. The X.509v3 also allows the use of the hash value of the subject itself, as in the case of executable authorization, but the option is currently not supported by PKIX.

Another feature of AC as defined by X.509v3 that is not currently supported by PKIX is privilege delegation. However, a mechanism is provided for making it possible for server to proxy an AC on behalf of the subject, by using an extension that contains a list of *predetermined* servers that are allowed to act as proxies.

As of this writing, the PKIX has published 11 Request for Comment (RFC) documents (9 proposed standard RFCs and 2 informational RFCs), with 19 Internet Drafts under reviews. The documents include specifications and guidelines for certificate profiles, operational protocols, management protocols, policy guidelines and

non-repudiation service components (time-stamp, data verification and data certification). An Internet Draft [14] being developed into an Informational RFC providing the history and overview of PKIX's various documents, is the place to start for the study of the PKIX standards. All PKIX documents are available on-line at [13].

## 2.6 IETF's SPKI Working Group

Another IETF working group focusing on the use of public key on the Internet is called Simple Public Key Infrastructure (SPKI). The justification for having two separate working groups under the IETF working on the same problem is that they have rather radically different ideas on what the best way to deploy digital certificates on the Internet should be. The differences span fundamental certificate theory, naming, certificate format, authorization and delegation models. The following quotes the SPKI working group charter [19]:

> "Many Internet protocols and applications which use the Internet employ public key technology for security purposes and require a public key infrastructure to manage public keys.
>
> The task of the working group will be to develop Internet standards for an IETF sponsored public key certificate format, associated signature and other formats and key acquisition protocols. The key certificate format and associated protocols are to be simple to understand, implement and use. For purposes of the working group, the resulting formats and protocols are to be known as the Simple Public Key Infrastructure, or SPKI.
>
> The SPKI is intended to provide mechanisms to support security in a wide range of Internet applications, including IPSEC protocols, encrypted

> electronic mail and WWW documents, payment protocols and any other application which will require the use of public key certificates and the ability to access them. It is intended that the Simple Public Key Infrastructure will support a range of trust models."

The opinion of the SPKI's proponents on subject identifier is that the use of human-readable globally unique name as identifier doesn't really work, especially in the cyberspace, where the interacting parties have never even met. Inescapable identifier is not feasible on a global scale since i) there is not likely ever to be a single globally accepted root naming authority, and ii) it would hinder personal privacy and organizational confidentiality requirements. It has also been shown that the names we regularly use do not have a global scalability; and that the common practice of using local names globally by users of PKI systems based on X.509, which has been going on despite the opposite claim specified in the standard text, is futile and dangerous. Also, the subjects to be identified by certificates are not limited to natural entities (e.g. human beings) but will also include artificial and logical entities (e.g. company, domain name, web servers, etc). A naming scheme that tries to cover all possible types of both natural and logical entities will be, to say the least, overly complicated, as evident in the X.509's AltName specifications.

Another fundamental argument is that it is *eligibility authentication* not *identity authentication* (a.k.a. identification) that is required in the authorization process. Identification is mostly required for accountability, e.g. payment for goods/services consumption or punishment for violation. Hence, identification should be part of the accounting process, not the authentication and authorization process.

For a comprehensive treatment on the short-comings of traditional PKI, please refer to [20, 4, 21]. In short, though, suffice it to say that any attempt to adapt X.509 for authentication/authorization/accounting in cyberspace is like retrofitting a ship

so that it can be driven on the road, or flown in the sky. We are better off with a car, or a plane.

Based on the requirements presented in [22], the SPKI theory [12] revisits the basic principal of public key cryptography and argues that a public key is globally unique and bound to its owner through the corresponding private key. Hence, the public key, or a collision-free hash thereof, can be directly used as the global identifier of the subject holding the corresponding private key. In the limit, a name, or rather a *nym* as defined in [4], is simply one of the many attributes that could be associated with a subject represented (i.e. identifiable) by its public key. As a result, there is neither a need for the distinction between public key certificate and attribute certificate nor for a trusted-third-party naming authority. An entity starts out its identity in cyberspace through its naked public key that accumulates attributes through certificates issued by the authorities of those attributes. Delegation of attributes is supported through the use of a boolean *delegation* field in the certificate asserting permission to delegate further. Thus, all basic SPKI certificates are of the form <issuer,key,attribute,delegation,validity>, or <key,attribute> for short.

However, if unique name is really needed, then they propose that "(the hash of) a public key (of the certificate issuer) followed by one or more names relative to that key," called Fully Qualified SDSI Name, can be used as the global identifier. For example, (name (hash sha1 |TKADlkjsDdf4jSkjjk2D5Ha8sktkKSFJ|) tep paul jose) is a Fully Qualified SDSI name of an entity locally named jose, defined by an entity locally named paul, which in turns has been defined by another entity locally named tep. tep is defined by the CA itself, which is globally identifiable by the SHA-1 hash of the its public key anchoring the name string above. A group (a.k.a. role) can also be defined by issuing multiple <key,name> certificates, all having the same name (the group name) but different keys, one for each group member. Note that

the local names need not always be general human names but also other forms, e.g. Canada's SIN number, Thailand's citizenship number. It is also possible to convert X.509 Distinguished Names into Fully Qualified SDSI names, if needs be.

Indirection (or delayed binding) through the use of <name,key> and <name,attribute> certificates is allowed in SPKI, but only if both certificates are issued by the same authority. Such indirection can be particularly useful when the life-time of the key is much shorter than the life-time of the name and the attributes.

Key life-time control is achieved by giving certificate limited life-time plus the use of non-overlapping CRLs and/or on-line revalidation. Short-lived certificates can also be used to reduce the needs for CRLs and on-line revalidation service, thereby reducing verification/validation overheads. It has also been shown in [23] that it is possible to organize the infrastructure in a way that entirely eliminates the needs for CRLs and revalidation.

For privacy protection, SPKI advocates the use of push-model for certificate path discovery. The subject is responsible for supplying the correct certificates chain to the verifier. Furthermore, it is recommended that each subject generates and uses multiple key pairs, each for different purpose. This can be done either by having one root key-pair for each application, or having one long-lived root key-pair delegates authorization to temporary key-pairs. The former provides the ultimate in privacy, while the latter provides simplicity. Furthermore, SPKI also suggests the use of *Certificate Reduction* service that reduces a chain of delegation certificates into a single certificate. Such reduction can prevents leakage of the organization's internal structure to the outside world.

Another interesting and potentially useful feature supported in SPKI is the *Threshold Subjects*, "specified by two numbers, K and N ($0 < K <= N$) and N subordinate subjects." A threshold subject (see Figure 2.3a) is considered valid if at least K out

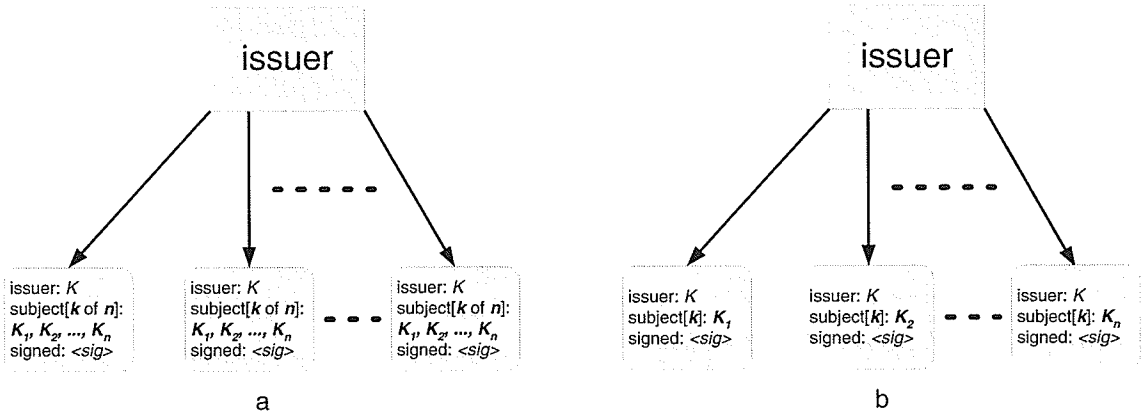


Figure 2.3: SPKI threshold subject (a) and open threshold subject (b)

of N subordinate subjects have been verified, similar to the case where a joint bank account opened by 7 (N) managers that requires signatures from at least 5 (K) managers in a withdrawal transaction. Recent work, however, has identified privacy and flexibility issues with the K-of-N threshold certificate. The proposed improvement, called *Open Threshold Certificate* [24] (see Figure 2.3b), eliminates the inclusion of N in the certificates and have the issuer signing a separate sub-certificate for each subject, all with the same certificate identification number and the threshold value K. The same work also introduces the *Identity Escrow Certificate* concept to provide an accountability mechanisms with minimal compromise of privacy protection.

To clarify the semantics of SPKI certificates, [12] presents a tuple reduction algorithm to process certificates and related objects to yield an authorization result. Additional rules for the translation of X.509, PGP, SSL and SDSI 1.0 certificates into compatible tuples that can take part in an authorization computation is also provided.

The SPKI working group had published 2 Experimental RFCs [22, 12] before the working group was discontinued due to lack of industry supports. There remained quite a few pieces of details missing before the SPKI concept becomes usable, in-

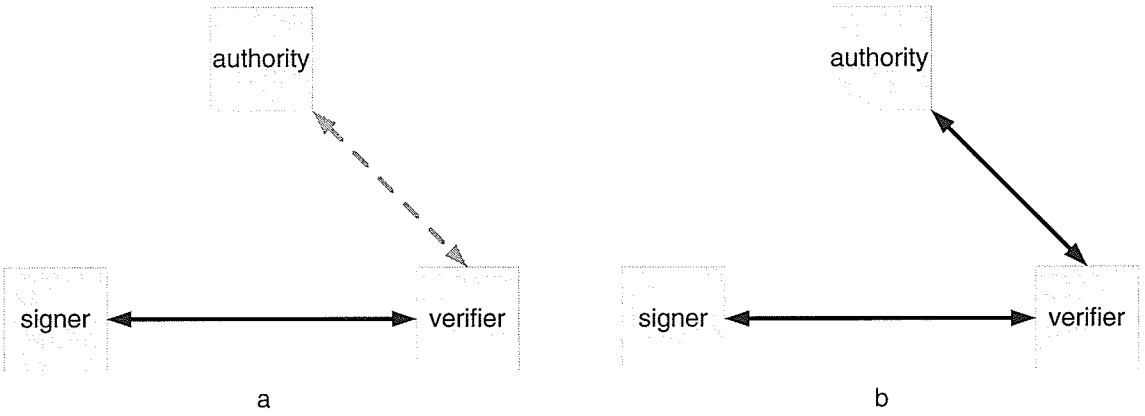


Figure 2.4: Off-line authorization (a) and on-line authorization (b)

cluding CRL format, on-line validation protocols, management protocols, reference implementations. Recent attempts to formalize and/or extend the semantics of SPKI certificates include [25, 26, 27].

## 2.7 Account Authority Digital Signature Model

Any electronic authorization process can be categorized as either *on-line* or *off-line*. An off-line session (or transaction) is one in which there exists only an iterative communication channel between the signer and the verifier and the verifier itself is not the authorization authority (see Figure 2.4a). In such scenario, the verifier only has access to the credentials supplied by the signer, plus, perhaps, some (negative) revalidation information obtained off-line some time in the past (the dotted line in Figure 2.4a). In an on-line session, however, there is also an interactive communication channel between the verifier and the authority (see Figure 2.4b). In this scenario, the verifier can directly request the authority to validate the signer's authorization status during authentication. This categorization forms the basis for the argument, summarized below, that leads up to the *Account Authority Digital Signature* (AADS) model for on-line authorization.

According to the AADS model creators, uses of certificates in *on-line authorization* is superfluous and results in unnecessarily, and increasingly, complex authorization systems (and the more complex the systems get, the harder (read *costlier*) it is to secure and maintain). The arguments goes as follows: since the verifier can validates the signer's authorization status interactively with the authority (termed *Account Authority*), the only credentials needed from the signer is the signer's public key and digital signature on the authorization request. This is analogous to the debit/credit payment transactions, in which the Bank is the account authority, the card holder the signer, and the merchant the verifier. The model also has a positive side-effect on privacy protection in that, the only information conveyed to the verifier can be on a need-to-know basis. The AADS model can be viewed as a PKI system without certificate.

Publicly available documentations on AADS are quite rare [28, 29]. The AADS model was also chosen by the ANSI for its X9.59 Standard Online Payment Protocol for Financial Industry. ANSI standard documents can be purchased from the ANSI on-line store [30].

However, since the AADS model addresses only on-line authorization, it should not be viewed as a total replacement for certificate-based process, termed *Certification Authority Digital Signature* (CADS) model. For example, there are some applications that operate on-line at some times and off-line at other times. Furthermore, the AADS model's requirement for on-line communication between the verifiers and the account authority also precludes distributed authorization and authority delegation allowed for by the CADS model. The two models should be seen as complementary, rather than mutually exclusive.

Let us take this perspective one step further by looking at the AADS model from another perspective. It is not too far-fetch to liken the account authority's response

to a dynamically-generated certificate which is valid only for a particular session as specified by the verifier. By introducing a session-bound type into the validity field in the certificate, as an alternative to the time-period type, we immediately open up a venue of possibilities in designing a PKI-based authorization model that provides the advantages offered by both the CADS and the AADS models. For example, in such model, it is possible for banks to issue credit certificates that allow for off-line authorization for small-value transactions but require on-line authorization for high-value transactions, minimizing both liability and on-line authorization cost.

## 2.8 Concluding Remarks

An introduction to public key cryptography and public key infrastructure and a review on the current state-of-the-art in public key infrastructure technology and standards have been presented. It has been shown that the X.509-based PKI model in widely use today has several fundamental flaws that may be too costly to correct by adapting the already all-too-complicated standards. However, there exist other alternative models like SPKI and AADS that address the shortcomings of the current model, but each of which is not adequate on its own. A new PKI model, based on some form of a merger between the SPKI and AADS, should be able to offer the best of both worlds and shall be pursued further in Chapter 4.

# Chapter 3

# Authorization Management Model

Before proposing a solution, I needed to understand the authorization management problem more clearly. Unfortunately, after months of searching, I could not find any reference that adequately models the problem. So, I took it as a task to develop a real-world authorization management model. The result of which is presented below.

## 3.1 System Participants

An authorization management system consists mainly of three types of participating entities (see Figure 3.1). These are the *Privilege Issuer*, the *Privilege User* and the *Access-control Monitor.*

Privilege Issuer is the entity responsible for issuing authorizations (a.k.a. *privileges* or *permissions*) to other entities (a.k.a. *subjects*). Authorization is usually issued in the form of *credentials* that are given to users for off-line authorization (e.g. driver's licenses, passports, bus/movie/concert tickets) or stored in an on-line validation database [1]. Privilege validity and authorization protocols will be discussed in details later in Section 3.4 and 3.5, respectively.

Historically, there are two types of issuers, i.e. *authority* and *trusted third party.*

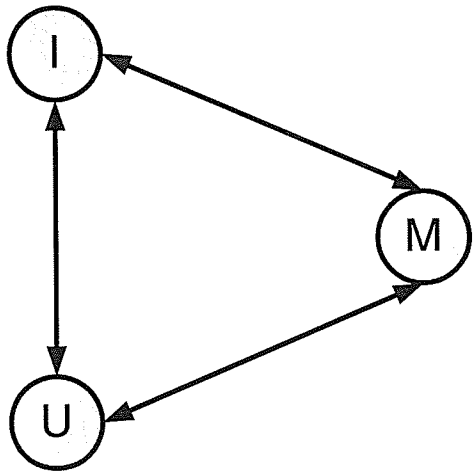


Figure 3.1: Authorization management system participants, i.e. Issuer (I), Monitor (M) and User (U)

An authority is an entity that actually have ownership and/or responsibility over a particular domain/region; these include a company's share-holders, a house's owner and a country's dictator. On the other hand, a trusted third party have no inherent power but is *trusted* by other entities involving in a transaction to help complete the transaction fairly and securely; these include banks, credit-card companies and attorneys [1].

Over the years, however, information-security companies (probably their marketing departments) have been creating a confusion over these two terms by using them interchangeably. For example, companies like VeriSign, whose business is issuing public-key certificates for a fee, like to call themselves *Certification Authorities* (CA). The fact is that VeriSign is not really an authority in any of the information (e.g. email addresses, people names, company names, internet-server hostnames, etc.) or any (implicit) authorizations (e.g. to create and use SSL sessions, to digitally sign and/or encrypt emails, etc.) contained in the certificates it issues. Some exceptions are its own company name and public signature key [1, 20]. Also, there are situations where an authority in one domain is used as a trusted third party in another totally

disconnected domain. For example, bars often rely on the date-of-birth printed on a customer's driver's license to verify his/her age before selling alcohol, thereby using the Department of Transport as a trusted third-party. I will revisit this issue again in Chapter 4.

Privilege User (a.k.a. *prover*, or *requester*) is the subject of an authorization that makes use of the received privileges. This means that the user has to obtain the privileges from the issuer somehow. The process that the user has to go through in order to obtain privileges from the issuer is often called *registration*. Registration procedures vary from application to application and are usually pre-defined by the issuer. For examples, obtaining a driver's license usually requires a proof-of-age, a 'pass' score on written and practical examinations, plus a payment of some fees. Obtaining a concert ticket, on the other hand, simply requires a (proof of) payment.

Usually, the subject of an authorization would be a single entity. However, there are certain applications where a privilege is shared amongst a number, say $n$, of entities in such a way that making use of the privilege requires at least a certain number, say $k$, of those entities to participate, where $1 < k \leq n$. The subjects of such shared privileges are called *Conjunct Subjects* [25]. Some examples of applications that employ conjunct subjects are joint banking accounts and nuclear missile launching systems.

Access-control Monitor (a.k.a. *verifier*, *acceptor*) is the entity responsible for ensuring that only authorized users are allowed to act on the privileges they have and no more. In some extremely small-scaled applications, one entity could be performing both as the issuer and the monitor. However, it is still safer to treat the two roles separately since they represent two logically-distinct tasks. A typical authorization management system would have at least one issuer, one or more monitor(s) and many users.

A real-world example of authorization management system is a multi-user operating system. The system administrator is responsible for issuing and managing user accounts on the systems and is therefore the issuer. A person who has an account on the system is a user. And the computer's operating system is the monitor. Another example is a driver-licensing system. The government's transport department responsible for issuing driver's licenses is the issuer. The people holding a driver's licenses are the users. And the traffic polices are the monitors.

Usually, there would be at least one monitor at every access-control point. However, in some applications, it may not be economically feasible to do so and the number of monitors would be far less than the number of access-control points. In those applications (e.g. driver's licensing) the users would be allowed to utilize the relevant privileges (e.g. driving cars on the roads) unauthenticated and the monitors (e.g. traffic polices) would passively/randomly *monitor* user's activities. The monitors would then enforce authentication only on the users with suspicious activities (e.g. an unmarked dark-windowed van driving late at night) and/or upon detection of violations (e.g. running a red light or going over a speed-limit).

Building an authorization management system involves defining and implementing pair-wise relationships and interaction conventions (a.k.a. *protocols*) between the issuer(s), the monitor(s) and the users in compliance with an established policy.

In general, the issuer, the monitor and the user exhibit the following notable properties and relationships:

- The monitor relies on the issuer's issued authorizations to be correct and current. However, the monitor is still the one making decisions to accept or reject a user's request.

- The issuer also relies on the monitor to correctly enforce the authorizations it has issued. Such reliance—which often implies *trust*—could be one-sided, as in the case of a bar using driver's licenses for age verification, or mutual.

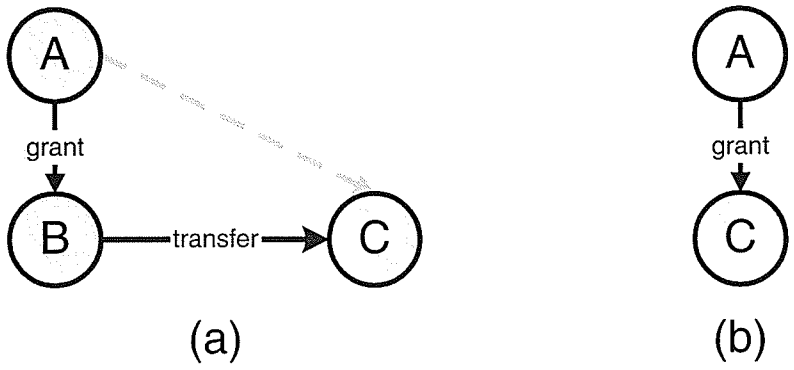


Figure 3.2: Privilege transfer: (a) before and (b) after

- The user relies on the issuer to issue credentials that will be accepted by the monitor.
- The user also relies on the monitor to honor the credentials issued to it by the issuer.

## 3.2 Empowerment and Privilege Propagation

In practice, there are two ways an issuer can empower its subjects with privileges: *granting* or *delegation*. Both imply that the issuer gives its subject a permission to do something. The difference is that with delegation the user is actually acting as an *agent*, performing some tasks authorized by the given permission on the issuer's behalf. This difference has some implications on accountability. This is because a user is solely liable for all actions it took with granted privileges. On the other hand, an issuer usually, at least partially, shares liabilities for the actions taken by its agents. Examples of granted privileges are driver's licenses, bus/movie/concert tickets and voting privileges. Delegations are used all the time in militaries, governments and corporations as a mean to divide duties and responsibilities amongst the organization members.

In an application that uses granting for empowerment, the issuer may choose to allow its subjects to transfer the granted privileges to other entities. As far as authorization and accountability is concerned, when a privilege is transfered from one entity to another entity, the second entity totally replaces the first entity from that moment onward. Bus/movie/concert tickets are examples of *transferable privileges*, while driver's licenses and voting privileges are examples of *non-transferable privileges*. A transferable privilege always stays transferable as long as it is still valid. The original issuer or a third party cannot stop a transferable privilege from being transfered further without revoking it.

Propagations of privileges can be visualized using directed graphs called *authorization chains*. Each node in an authorization chain represents an entity and each arc represents the propagation of privileges between two entities. An entity with only out-going arc(s) and no in-coming arc is the *root issuer* (or *root authority*). An entity with only in-coming arc(s) and no out-going arc(s), i.e. the leaf node, is an *end subject* (or *end user*). An entity with both out-going and in-coming arcs acts both as an issuer and a subject. Figure 3.2a shows an authorization chain where entity $A$ granted a transferable privilege to entity $B$, which then transfered the privilege to entity $C$; the logical result of which is shown in Figure 3.2b. Since transferable privilege always stays transferable, later on, entity $C$ might decide to transfer the privilege to another entity as it sees fit, and so on and so forth.

In an application that uses delegation for empowerment, the issuer may choose to allow its agents to re-delegate (a subset of) the delegated privileges further, thereby acting as an issuer. Unlike privilege transfers, though, the entity issuing a delegation does not loose the delegated privilege and any accompanying accountability. A single authorization chain of delegations that begins with a root issuer and ends with a user is called a *delegation chain*. Figure 3.3a shows two simple delegation chains, i.e.

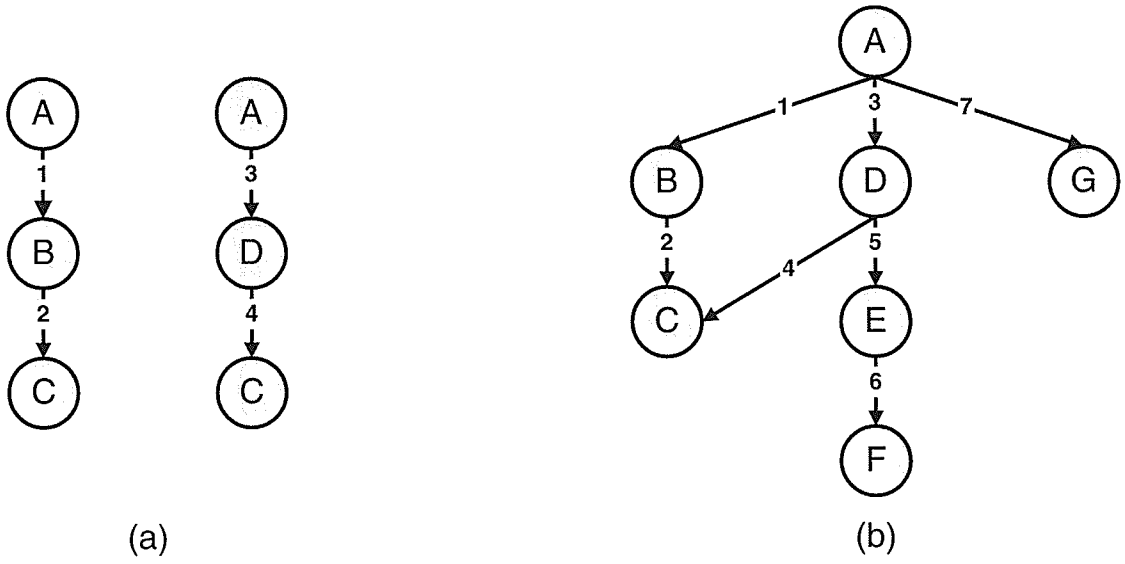


Figure 3.3: Examples of (a) delegation chains and (b) delegation graph

$A \rightarrow B \rightarrow C$ and $A \rightarrow D \rightarrow C$. Multiple delegation chains that represent delegations of the same set of privileges from the same root issuer can be union-ed to form a *delegation network*, as shown in Figure 3.3b.

Note that each lower level of delegation in a delegation chain can contain only a subset of or at most the same privilege(s) as that of the upper level of delegation. An entity may not issue a privilege that it does not have. For example, in the $A \rightarrow B \rightarrow C$ delegation chain from Figure 3.3a, if entity $A$ has delegated the permission to 'read file x and delete file y' to entity $B$ via arc #1 then entity $B$ may not delegate the 'write file x' or 'read file y' permissions to entity $C$ via arc #2. However, it is acceptable for entity $B$ to delegate 'read file x' or 'delete file y' or 'read file x and delete file y' to entity $C$.

In some applications, it might be desirable and useful for the root issuer to be able to grant a *delegateable privilege*. For example, the Ministry of Trades and Mines grants a copper-mining permission to (fictional) National Miners Corporations Limited, which in turn delegates the grant to one of its subsidiary, say Manitoba Miners

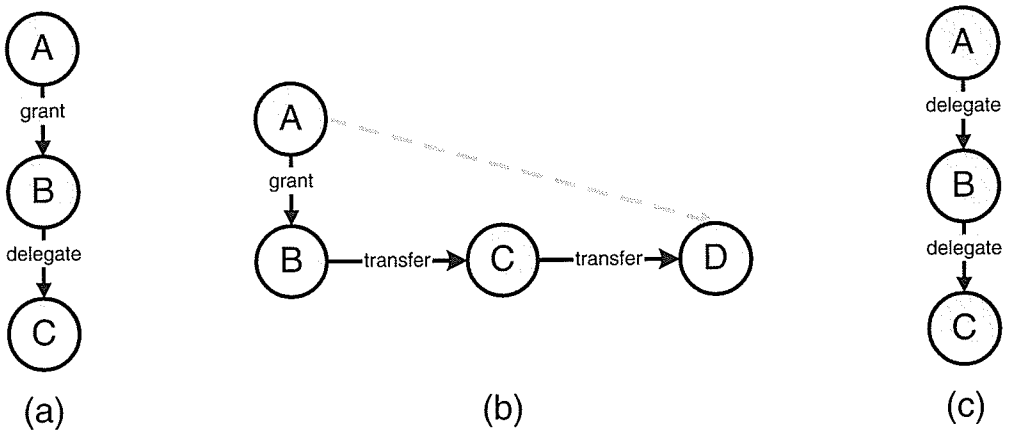


Figure 3.4: Legal privilege propagations

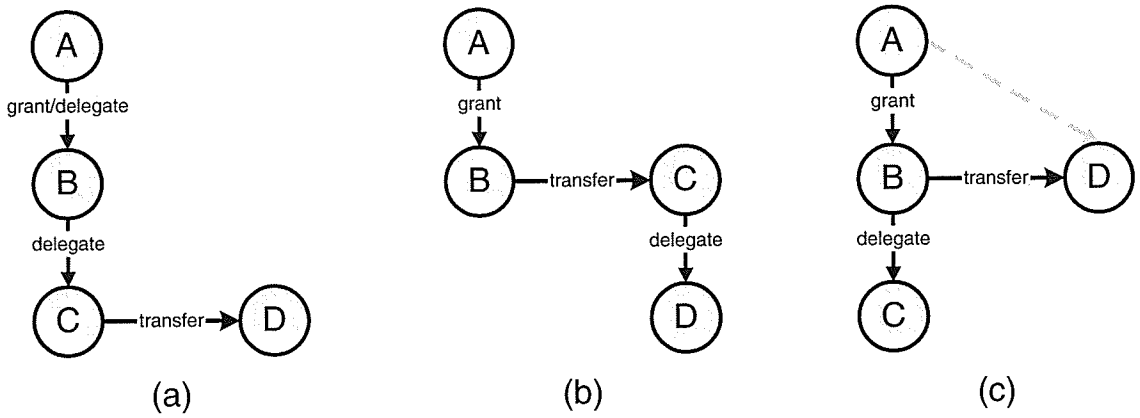


Figure 3.5: Illegal privilege propagations

Incorporated, to take care of copper mining within the province of Manitoba. Manitoba Miners Incorporated may then delegate part of the job to another company if that was allowed by National Miners Corporations Limited.

Figure 3.4 shows the three forms of privilege propagations discussed so far.

On the other hand, transfer and delegation are mutually exclusive. They do not coexist in the same authorization chain. This is because delegated privileges cannot be transfered (see Figure 3.5a). This should be rather obvious, since delegation means the issuer and the subject are sharing the privileges, as well as accountability. If the

agent wants to quit then it cannot just transfer the delegated privilege to another entity of its choosing. It has to inform the delegating entity first so that it can finds a new and appropriate agent. Also for the same reason, transfered privileges cannot be delegated (see Figure 3.5b). Furthermore, we can infer from the same argument that transferable privileges are not delegateable and delegateable privileges are not transferable (see Figure 3.5c).

Therefore, the forms of propagations shown in Figure 3.5 are not allowed.

## 3.3 Access-control Conventions

Access-control conventions refer to the conventions for communicating and interpreting of authorization information between the issuer and the monitor. A number of conventions have been proposed and used, but I will focus on the two most widely adopted conventions: Role-based and Capability-based Access Control.

With *Role-based Access Control* (RBAC) [31, 32], participants in a given domain have to agree on a naming convention and a name-space that provides a consistent mapping between (groups of) privileges, i.e. *roles* and their respective identifiers, i.e. *nyms*. Privileges associated with each role may *partially* overlap with those of other roles. The issuer would then assign one or more role(s) (e.g. manager, purchaser, accountant, clearance level-2) to its subjects. A monitor would maintain an Access Control List (ACL) which maps between the various roles and permissions, e.g. a 'manager' can enter restricted area, an 'accountant' can access the company's financial records. A user wishing to gain access to some resources (e.g. enter a restricted area, log-in to the corporate network) or perform some tasks (e.g. issue a purchase order) would have to convince (i.e. authenticate to) the monitor that it really holds (one of) the role(s) that maps to the appropriate permission.

A key characteristic of role-based authorization is that when the privileges asso-

ciated with a particular role are changed (by modifying the ACL) at any time, the changes would immediately effect all entities associated with that role. This is proven to be both beneficial and problematic. The obvious benefit is that privileges can be conveniently assigned to and/or revoked from a group of users having the same role without having to individually revoke and reissue all the respective credentials. This helps easing the authorization management in an environment where a large number of users can be naturally grouped into a relatively smaller number of roles. Furthermore, each participating monitor could be programmed to treat each role differently from other monitors according to its local policy. Therefore, enabling distributed authorization policies. The trade-off is in the lost of granular control on individual users.

In a variation of RBAC where each role has a one-to-one mapping to one user (a.k.a. *discretionary access control*), the ACL would simply contain individual records of all users and their corresponding permissions. An example of this is a conventional multiuser operating system.

*Capability-based Access Control* (CBAC) moves the burden of ACL management to the issuer. In this case, participants have to agree on a capability description language to communicate privileges between the issuer and the monitor(s). A monitor is simply programmed to honor any (or a certain set of) privileges issued by the trusted issuer(s). The issuer would then include prescriptive capabilities (e.g. 'can issue purchase order up to $2,000', 'can access http://host.domain.com/path/', 'can drive motorcycle') in the credentials issued to users. This makes for very simple implementation of the monitors. It also offers a very fine granularity of control, at the expense of additional responsibilities (and, perhaps, accountability) on the issuer. Examples of CBAC systems are credit- and debit-card payment systems, various licensing systems and bus/movie/concert ticketing systems.

## 3.4 Validity and Deactivation

Every issued privilege has a limited validity, either in the number of times it can be used or within a time period. For example, a concert ticket can be used only once at a specified performance on a specified date and time; a Manitoba driver's license is valid for one year; a Thai passport is valid for up to ten years; a Winnipeg Transit's week-day bus pass is good from Monday to Friday on the specified week; a life-time membership at a sports club is valid as long as the member is alive and the club is operating; a cable-television service is active as long as the customer pays the monthly bills on time. Limited validity ensures that any abused/misused/unused privilege would eventually and automatically expire. However, there are applications that require the ability to actively deactivate privileges before its *expiration* time, such as when credentials are lost/stolen, when fees are not paid on time, when drivers violate traffic laws, or when employees quit (or are laid-off/fired) prematurely.

*Premature deactivation* of privileges may be *permanent* or *temporary*. Permanent deactivation is called *revocation* and temporary deactivation is called *suspension*. Once a privilege is revoked, the only way for a user to receive the privilege again is to go through the registration process again and have the issuer reissues the privilege (Of course, if the particular user is permanently black-listed by the issuer then reissuing may not even be possible). On the other hand, a suspended privilege can be reactivated again either when the specified suspension period is over or when the user satisfies some conditions, e.g. paying fine or outstanding bills. Sometimes, suspended privileges might end up being revoked, such as when the customer still did not pay old bills after one month of suspension. In some applications, the issuer may simply choose to skip suspension and employ only revocation because the cost associated with reissuing is low enough relative to the cost associated with enforcing suspension.

Since every authorization—be it through granting or delegation—involves bilateral

agreements between an issuer and a subject, a decision to prematurely revoke a privilege could come from either entity. If the issuer (or an issuer's designated agent) initiates the revocation, then it is called *issuer-initiated revocation.* Otherwise, if the user voluntarily gives up its privilege, then it is called *self-initiated revocation.*

When premature deactivation is applied to a part of a delegation chain, all relying delegations are automatically deactivated as well. This is called *cascade revocation* [32]. For example, if entity $A$ in Figure 3.3b decided to revoke the delegation arc #3, then delegation arc #4, 5 and 6 would also be revoked as well, leaving only the $A \rightarrow B \rightarrow C$ and $A \rightarrow G$ chains in the graph.

As far as validity is concerned, there are two types of credentials: one-time credentials and reusable credentials. *One-time Credentials* can be used only once, with the validity specified in terms of a unique description of an event/session where the credential is valid. Or it could be specified that only single usage is allowed, possibly within a given time period. Examples of one-time credentials include movie/concert/bus tickets and supermarket discount coupons. *Reusable Credentials*, on the other hand, can be used more than once. In this case, validity may be specified as the number of uses allowed, the time period in which unlimited number of uses is allowed, or a combination of both. Examples of reusable credentials include monthly bus passes, driver's licenses, passports and membership cards.

## 3.5 Validation Protocols

Prior to implementing an authorization system, participants need to agree on the protocol(s) to be used for validating privileges. There are many protocols in use for this purpose, but they can all be classified as either off-line authorization or on-line authorization.

*Off-line Authorization* is used when the issuer is not available to validate the user's

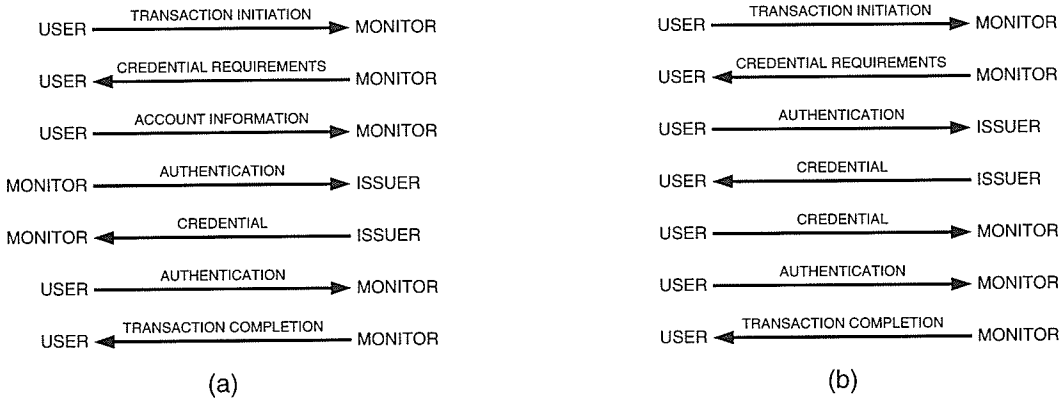


Figure 3.6: Examples of on-line authorization protocols; (a) credit-card payment protocol and (b) debit-card payment protocol

privilege status during the authorization process. In other words, the monitor has to rely on the credentials issued by the issuer at some time in the past. The user is always responsible for obtaining appropriate credentials beforehand, normally as part of the registration process. Credentials issued for off-line authorization may be one-time credentials or reusable credentials.

*On-line Authorization*, on the other hand, requires that the issuer—or its agent, called *validator*—be available to validate the user's privilege status upon requests. Using such protocols, the monitor is assured that the supplied authorization information is correct and current at the time when the session/transaction takes place. Credentials issued for on-line authorization are always one-time credentials since they are bound to specific sessions/transactions at the time of issuance. However, the entity responsible for obtaining credentials from the issuer may either be the user or the monitor, as shown in Figure 3.6.

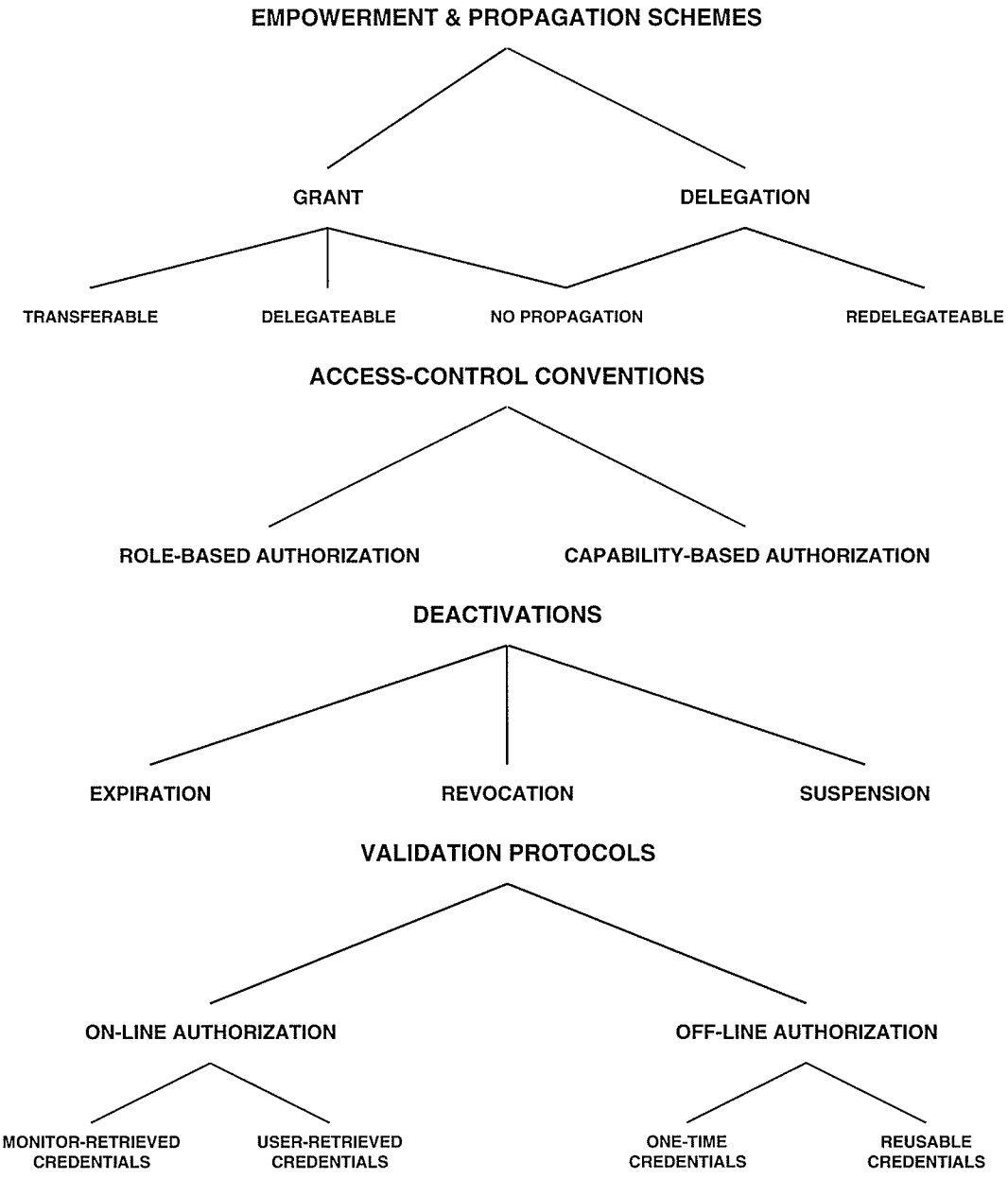


Figure 3.7: The authorization management model

## 3.6 Summary

By deriving from the discussions presented in this Chapter, the authorization management problem can be reduced into the Authorization Management Model shown in Figure 3.7. All key aspects of the problem, i.e. empowerment and privilege propagations schemes, access-control conventions, validity and deactivation and authorization protocols, are represented in the model. I will use this model (hereafter referred to as *the model*) as the basis in developing the Authorization Management Framework in the next chapter.

# Chapter 4

# Authorization Management Framework

Now that the taxonomy of the problem has been established, it is time to consider the solution. First of all, since the problem involves dealings with human behavior and many out-of-scope but related problems, a list of assumptions are given to narrow down the scope of the problem. The solution is then presented in two parts: the *Conceptual Framework* and the *Infrastructure Implementation Guidelines*, which together comprise the *Authorization Management Framework* (AMF).

## 4.1 Assumptions

**Assumption 1** *Protection of private information is achieved through the mechanisms provided by the design and implementation of supporting infrastructure and applications, in conjunction with cooperation of the educated individuals and regulatory protection.*

Protection of privacy is a civil-right issue which cannot be addressed by technology alone. Adequate regulatory framework has to be in place to protect individuals by

ensuring accountability on violators. Such protections should include, but are not limited to:

- Each entity is responsible for generating and assuming absolute ownership and control over its signature key-pairs and the device on which the private signature key is generated and stored [33, 34].

- Escrow of private signature key is a violation of privacy and must be strictly prohibited.

- Escrow of a key-holder's identity should be optional and must require the accountability requirements of relying parties and mutual consent of the key-holder.

  This is because identification of session/transaction participants is part of accounting, rather than authentication and/or authorization [24].

- Uses of personal information by any third-party for any purpose other than accountability (e.g. marketing) must require mutual consent of the identified individuals.

**Assumption 2** *Resource requirements (e.g. processing power, storage space) for key-pair generation is manageable in the way that permits key-holder to casually generate multiple key-pairs for use in different applications and/or circumstances.*

**Assumption 3** *A key-pair is used for either digital signature or data encryption only, and never for both.*

**Assumption 4** *Time synchronization among system participants is assumed to be resolved and is beyond the scope of this work.*

## 4.2 The Conceptual Framework

The framework proposed here is primarily based on and adapted from the SPKI theory [12] and the AADS model concept [29, 28] reviewed earlier. Conceptual solutions are presented in the form of *propositions*, along with any outstanding *challenges* to be resolved through the infrastructure or other means.

### 4.2.1 System Participants

The first step is to figure out how to electronically represent the various participating entities within the system—be it the Issuer, the User, or the Monitor. In this regard, the SPKI theory offers the simplest and the most robust basis for the use of public-key cryptography to represent and identify entities in cyberspace.

**Proposition 1** *A private signature key represents a digital persona of the entity controlling the key [12].*

**Challenge 1** *Ensuring that there is only one copy of a particular private signature key.*

**Challenge 2** *Ensuring that each private signature key is under total control of the key-holder, and no other entity, at all times.*

It is worth noting that the two challenges above are the fundamental issues that any public-key cryptography system must address. Theses will be addressed as a part of the infrastructure presented later in Section 4.3.

**Proposition 2** *(The collision-free hash of) a public signature key is the identifier of the digital persona represented by the corresponding private signature key [12].*

Thus, all entities participating in a session/transaction—be it natural entities like human beings, or artificial entities like organizations or software agents—are uniformly represented and identified by their signature key-pairs.

However, a human entity may, and usually will, need multiple digital personae that represent different roles one may performs.

**Proposition 3** *An entity differentiates among its multiple personae by employing a different key-pair for each persona [21, 12].*

**Challenge 3** *Making generation, registration, utilization and maintenance of multiple key-pairs manageable for the human key-holder.*

Although the AADS model's proponents [29, 28] argue that the Issuer-defined *account number* is the most robust identifier, I argue that it is not desirable for two reasons. First, account numbers represent *inescapable permanent identifiers* assigned under total control of the Issuer, which may be used as an aid in the collection of User's activities to generate a privacy-invasive dossier. Second, different entities may be sharing the same account, as in the case of corporate account and co-signing account. There is no way of differentiating those entities for *accountability*. On the other hand, using public signature keys as identifiers means that the key-holder has better control over its digital persona and yet can still be held accountable for its actions. Also, the Issuer always has to maintain a mapping between each public signature key and its associated account number anyway. Keeping such mapping internal greatly reduces the risk of privacy invasion from third-party Monitoring the User's activities.

**Proposition 4** *A digital signature on a statement serves as an assertion on the authenticity and integrity of the statement, in the signer's opinion.*

Or, in SPKI terminology "A Principal is said to 'speak' by means of a digital signature. ... The Principal is said to 'speak for' the Key-holder." [12]. The *Principal* in this case is the key-holder's signature key-pair, identified by the public signature key. From this point on, I will use the term *'principal'* to refer to (the hash of) an entity's public signature key.

Since all entities are represented and identifiable by their signature key-pairs, the most straight-forward way for an Issuer to assign privileges to its subjects is through the subjects' principal using electronic credentials. Such credentials are called *"authorization certificates"* in SPKI terminology. The syntax of a basic authorization certificate is ($K_I$ says authorize($K_S$, *Privilege*)), where $K_I$ is the Issuer's principal and $K_S$ is the subject's principal.

**Proposition 5** *An electronic credential is represented in the form of an authorization certificate, digitally signed by the Issuer's private signature key, authorizing a certain (set of) privilege(s) to a subject.*

In case where the Issuer needs to issue privilege(s) to conjunct subjects, the Open Threshold Subject concept [35, 24] reviewed in Chapter 2 offers significant improvements, both in terms of privacy enhancement and simplicity, over the original threshold subject proposed in the original SPKI theory. Also, The SPKI threshold subject specifies the threshold value $k$ as a positive integer, where $k \geq 1$. However, a threshold subject with the threshold value of 1 have no added-value since it is equivalent to a singular subject.

**Proposition 6** *Use Open Threshold Subject [35, 24], which specifies a threshold value ($k > 1$) and a threshold scheme identifier, when issuing privileges to the conjunct subjects.*

So, each threshold subject $K_{S_x}$ would receive a separate authorization certificate with the subject field in the form (($\langle k \rangle$ $\langle$scheme-id$\rangle$) $K_{S_x}$). By this the Issuer indicates

that each subject with the same ⟨scheme-id⟩ receives $\frac{1}{k}$ fraction of the right to use the specified privilege. This gives the Monitor just enough information to make its access-control decisions without impeaching on the privacy of other subject(s) not participating in the particular transaction/session.

### 4.2.2 Empowerment and Privilege Propagation

The SPKI theory only deals with delegation. In some places, their documents [12, 36] even use the term *grant* almost interchangeably with *delegate* and *transfer*. However, as the authorization management model presented earlier suggests, we need to differentiate between *granting* and *delegation*. So, the following three modifications to the SPKI theory in this regard are proposed:

1. When issuing a privilege, the Issuer must indicate in the credential the intended type of empowerment, i.e. granting or delegation.

2. When granting a privilege, the Issuer must indicate in the credential whether transfer or re-delegation is permitted.

3. When delegating a privilege, the Issuer must indicate in the credential whether re-delegation is permitted.

**Proposition 7** *Replace the SPKI authorization certificate's boolean* delegation *control field with an* Empowerment *field which has two sub-fields. Fist, the* empowerment-type *would indicate the type of empowerment: granting or delegation. Second, the* propagation-control *would indicate if either transfer or re-delegation is allowed.*

So, the syntax of an authorization certificate becomes either ($K_I$ says grant($K_S$, *Privilege*, $P$)) or ($K_I$ says delegate($K_S$, *Privilege*, $P$)), where $P$ would be set to either *no-propagation*, *transferable*, or *delegateable*.

From this, a delegation chain can be represented by a chain of delegation certificate(s), with the root Issuer issuing the (first) certificate and the User as the subject of the (last) certificate. Similar to the original SPKI theory, all certificates in a delegation chain, except for the last one, must have propagation-control set to *delegateable*.

It has been shown that the only way to enable privilege transfer using public-key cryptography is to *totally* transfer all the privileges associated with one principal to another principal [37]. So, I propose a new type of certificate called *transfer certificate*.

**Proposition 8** *A principal transfers its privilege to another principal by i) issuing a transfer certificate, then ii) immediately destroying the private signature key that corresponds to the certificate-signing principal [37].*

So, the syntax of a transfer certificate is ($K_I$ says transfer($K_S$, $(*)$)), where $K_I$ is the transferring principal and $K_S$ is the receiving principal, and $(*)$ means all privileges.

**Challenge 4** *Making sure that the private signature key is destroyed immediately after signing a transfer certificate.*

From this, a granted privilege can be represented by a granting certificate, perhaps followed by either a delegation chain, if its propagation-control was set to *delegateable*, or a (chain of) transfer certificate(s) if its propagation-control was set to *transferable*.

### 4.2.3 Access-control Conventions

The SPKI theory directly supports[1] CBAC through a capability-description language called *authorization tag*, that can be used to expressively define a specific privilege or

[1] on second thought, *advocates* might be more accurate

a group of privileges (e.g. wild-card, set, prefix, and range). However, developers are free to define their own extended syntax and semantics to fit specific applications.

**Proposition 9** *Use the SPKI's authorization tag to define capabilities in authorization certificates under capability-based access control systems. Such certificates are called 'capability certificates'.*

So, the syntax of a capability certificate becomes ($K_I$ says authorize($K_S$, *tag*, $P$)).

On the other hand, Section 4.3 of the SPKI theory [12] presents an argument against conventional RBAC by pointing out the cost and complexity involved in maintaining an ACL at the Monitor. The underlying argument is that CBAC is a preferred system for SPKI working group members.

However, as the model pointed out, RBAC has certain useful properties that are missing in CBAC. SPKI v2.0—a.k.a. SPKI/SDSI—works around CBAC's shortcomings by allowing the use of SDSI name as subject of the capability certificate and using name certificate(s) to map a name to principal(s). The syntax of a name certificate is ($K_I$ says bind(*name*, *subject*)), where the *subject* field is either a principal or another SDSI name. In an essence, such capability certificates are individual ACL entries signed by the Issuer and distributed to the Users. Each would then be presented to the Monitor along with the name certificate upon authentication. The Monitor's ACL is reduced down to (a list of) the root Issuer's public signature key(s), as in conventional CBAC. Such indirection, argued the SPKI theory, provides the benefits offered by RBAC without incurring the ACL management overhead on the Monitor.

Indirection through name(s) is optional for capability certificates. And as Section 8 of the SPKI theory cautions, the capability certificate and its associated name certificate should come from the same Issuer to avoid cross-talks between name-spaces belonging to different principals. For example, a capability certificate ($K_I$

says grant('$K_S$ *manager*', (*read file* x), *no-propagation*)) implicitly allows principal $K_S$ to propagate the privilege to *read file* x to any other principal at will by issuing name certificate(s) that map the name '$K_S$ *manager*' to a principal or even other name(s) in some other name-space(s). Howell & Kotz performed a formal analysis on the semantics of SPKI names and suggested that "... principals only define the first level of names in their name-spaces; all other names are consequences of chained first-level name definitions." [25]. So, care should be taken when using foreign names as subjects of capability/name certificates or when mapping local names to foreign names with name certificates.

Names could be quite powerful, and sometimes even dangerous, when used in threshold subjects. For example, the certificates ($K_I$ says grant((2 $SID_1$) '$K_I$ *manager*', (*read file* x), $P$)) and ($K_I$ says grant((2 $SID_1$) '$K_I$ *supervisor*', (*read file* x), $P$)) could have two meanings. It could be that mutual consent from (i.e. co-signing of request) one principal bound to the name '$K_I$ *manager*' and another principal bound to the name '$K_I$ *supervisor*' is required in order to *read file* x. Otherwise, it could be that one principal bound to both names through, perhaps some combination(s) of, name mapping(s), re-delegation(s), or transfer(s) is required. In general, though, it would be safer to refrain from using names in threshold subjects altogether.

**Proposition 10** *The subject of a capability certificate may either be a principal, an Issuer's local name, or a subject's local name. If name is used, it must be reducible to a principal through name certificate(s).*

So, the syntax of a capability certificate becomes ($K_I$ says authorize(*subject*, *tag*, $P$)), where the *subject* is either a principal or a local name.

In my opinion, the SPKI theory's argument against RBAC, though compelling at a first glance, missed a few subtle but crucial points. Signing individual ACL entries and distributing them as certificates does not really eliminate ACL management;

it simply relocates the work away from the Monitor and put additional work on the Issuer. Instead of just maintaining the mappings of *either* (*role* $\mapsto$ *principal*) or (*privilege* $\mapsto$ *principal*), the Issuer now has to maintain the mappings of *both* (*privilege* $\mapsto$ *role*): ACL entries, and (*role* $\mapsto$ *principal*). Also, by distributing ACL entries as certificates to Users, many of which will be duplicated, changing the privileges associated with a role means revoking all old capability certificate associated with the role, informing all relevant Users about it, issuing the new certificate, and then distributing it to all relevant Users. With conventional RBAC, all that is needed is to update the ACL entries at the relevant Monitors, the number of which is typically far smaller than the number of Users. These are some of the trade-off that have to be considered before picking one convention over the other. So, to satisfy the authorization management model, a way to accommodate RBAC, as well as CBAC is needed.

Ninghui Li [26] analyzed SPKI/SDSI *linked local names* by interpreting them as distributed groups and proposed a name-resolution logic program for SPKI/SDSI. Deriving from his logic program, Li motivated the use of local names as *distributed roles* and proposed the following restricted version of SPKI to mimic RBAC.

- The Monitor maintains ACL entries of the form ('$K_I$ *name*', *tag*), mapping Issuer's local names to privileges.
- The Issuer issues a name certificate to each subject, binding a name '$K_I$ *name*' to either the subject's principal $K_S$ or a subject's local name '$K_S$ $name_2$'.
- If the subject is $K_S$, then principal $K_S$ becomes a member of the role '$K_I$ *name*'.
- If the subject is '$K_S$ $name_2$', then every principal defined by '$K_S$ $name_2$' becomes a member of the role '$K_I$ *name*'.

The principal form of the subject allows principal $K_S$ to adopt the role '$K_I$ *name*', but not to re-delegate it further. On the other hand, the local-name form allows

principal $K_S$ to re-delegate the role '$K_I$ *name*' by issuing name certificate(s) that bind '$K_S$ $name_2$' to other principal(s), other local name(s), or back to $K_S$ for its own use. This, claimed Li, essentially implements SPKI's boolean re-delegation control without the delegation-control field.

Li's framework is quite attractive because it is both simple and practical. However, it needs a bit of adaptation to accommodate granting and transferring in my model. So, I propose a new type of certificate called *role certificate*. The syntax of which is ($K_I$ says authorize(*subject*, $R$, $P$)), where $R$ represents a role using an Issuer-defined name.

**Proposition 11** *Use the Issuer's local names to define roles in authorization certificates under role-based access control. Such certificates are called 'role certificates'. The subject of a role certificate may either be the subject's principal or, if re-delegation is allowed, a subject's local name.*

As examples, the certificate ($K_I$ says grant($K_S$, '$K_I$ *manager*', *no-propagation*)) means that principal $K_I$ grants the role '$K_I$ *manager*' to principal $K_S$. On the other hand, the certificate ($K_I$ says delegate('$K_S$ *manager*', '$K_I$ *manager*', *delegateable*)) means that principal $K_I$ delegates the role '$K_I$ *manager*' to principal $K_S$, and also permits $K_S$ to re-delegate it to other principals through '$K_S$ *manager*'. In either case, the Monitor's ACL would include an entry that maps '$K_I$ *manager*' to actual privilege(s).

Threshold subjects could be a bit tricky when used in re-delegateable role, too. For example, the certificates ($K_I$ says grant((2 $SID_1$) '$K_{S_1}$ *supervisor*', '$K_I$ *manager*', *delegateable*)) and ($K_I$ says grant((2 $SID_1$) '$K_{S_2}$ *supervisor*', '$K_I$ *manager*', *delegateable*)) means that it would require either A) mutual consent from (i.e. co-signing of request) one principal which received a delegation of the role '$K_{S_1}$ *supervisor*' (or $K_{S_1}$ itself) and another principal which received a delegation of the role '$K_{S_2}$ *super-*

*visor*' (or $K_{S_2}$ itself) or B) one principal which received delegations of both roles in order to *read file x*.

It is worth noting here that delegations of roles are always total, while capabilities may be partially delegated. Transfers are always total in both cases, though.

### 4.2.4 Validity, Deactivation and Validation Protocols

The SPKI theory allows two types of validity information to co-exist in the authorization and name certificates, i.e. the validity interval and the on-line tests. Validity intervals are specified as the 'not-before' and 'not-after' dates. Three kinds of on-line tests information are specified: certificate revocation/revalidation list (CRL) location(s), on-line validation service location(s) and replacement certificate issuing service location(s). However, I disagree on several counts as follows:

1. Short-lived certificate issuing service is supposed to be part of an off-line validation protocol. Since the User is always responsible for fetching the certificate and the certificates are reusable within its short life-time, they clearly fall within the domain of off-line authorization according to my model.

2. CRL is also supposed to be part of an off-line validation protocol. Although a CRL offers more *recent* information than certificate held by the User, such information is still not *current*. This is because CRLs are periodically issued with non-overlapping validity interval. So, even the information contained in the latest CRL is already outdated by the time the Monitor receives it. Furthermore, I rather agree with Rivest's argument that CRLs are not necessary [23]. There are several alternative mechanisms that can be used to ensure recency without using CRLs, e.g. short-lived certificates and on-line validation. Specifically, the two alternatives provide higher quality of recency information with lower operational complexities and costs than CRLs.

3. Rivest also proposed that the certificate's validity interval should include a 'maybe-after' date, in addition to the 'not-before' and 'not-after' dates. This enables the Issuer to indicate that the information in the certificate is definitely valid between 'not-before' and 'maybe-after'. It is up to the Monitor to decide whether or not to accept the certificate between 'maybe-after' and 'not-after', depending on the risk involved.

On the other hand, I do not agree with Rivest's assertion that the Monitor (or 'acceptor', in Rivest's terminology) is always the only one responsible/liable for its access-control decisions. In this respect, I agree with the SPKI theory that there may be situations where the Issuer and the Monitor must share the risk together and even where the Issuer is solely responsible.

The AADS concept [29, 28] argues that certificates are superfluous in on-line authorization. This is because the Monitor always contacts the Issuer to validate a User's privilege anyway. Indeed, that argument certainly applies to a centralized authorization system, where there is only one root Issuer and propagation is never allowed. However, upon encountering a delegation chain, the Monitor may not know in advance where and how to contact some or all Issuers in the chain except the root Issuer. So, a non-root Issuer needs a mechanism to reliably relay its on-line validation service information to potential Monitor(s).

The simplest way to achieve this is to put such information in certificates and let the Users supply them to the Monitor. In a *pure* on-line authorization system, certificates issued to Users would include just the Issuer field, the subject field and the on-line validation service location(s) in the validity field. This is analogous to the "account-signature card" issued by bank to its customers. Using this information, the Monitor would contact each Issuer (or a specified agent) in the chain, requesting for an authorization certificate—i.e. the credentials—that contains the actual privilege

information. As the model suggested, a certificate issued by the on-line validation service is always a one-time certificate. Its validity field must contain a single-usage restriction with a unique session/transaction *nonce* generated by the Monitor. Similar procedure applies where the User is responsible for obtaining the credentials.

In any case, however, the on-line validation process, whether performed by the Monitor or the User, always produces a (chain of) certificate(s) that contain only the off-line portions of the validity fields.

On the other hand, the Issuer might pre-specify the unique nonce in one-time certificates issued for off-line authorization. In this case, the Monitor would have to store the Issuer-defined nonce of used certificates, perhaps until the 'not-after' date and thereafter rejects all requests that use certificates containing used nonce. Since digital certificates require virtually no physical space, reusable credentials with a limit on the number of usages, say $M$, could be implemented simply by issuing $M$ one-time certificates.

**Proposition 12** *The validity field of an authorization and name certificate may include a validity interval—i.e. not-before, and/or maybe-after, and/or not-after—and/or either a) a single-usage restriction in form of a session/transaction nonce, and/or replacement (short-lived) certificate issuing service location(s), or b) on-line validation service location(s). An empty validity field means that the certificate is always valid.*

Validity is much simpler in the case of transfer certificates, though, since all privilege transfers are final.

**Proposition 13** *The validity field of a transfer certificate must includes only the not-before date.*

**Proposition 14** *In an on-line authorization system, deactivation is achieved by either removing (i.e. revoke) or disabling (i.e. suspend) User's account in the Issuer's*

*database, which would cause future validation requests to fail. In an off-line authorization system, short-lived certificates should be used and deactivation is achieved by stop issuing replacement certificates.*

There are two special cases, though; when a private signature key is lost or compromised. If the key is lost with no chance of being compromised (e.g. destroyed) then the key-holder could simply generate a new signature key-pair and register the new public signature key with the issuer (and request for new certificate(s)).

However, if the key-holder or the Issuer believes or suspects that the private signature key may have been compromised then Rivest [23] proposed the use of a "*suicide note*", signed in advance by the private signature key and stored off-line in a secured place. A neutral "*suicide bureau*" will, upon request, issue a "*health certificate*" certifying that a key-pair has not been compromised, unless it has received a suicide note signed by the corresponding private signature key. This line-of-thoughts deserve further analysis and perhaps could be improved upon, but will, for now, leave it for the future works.

The SPKI theory suggested using the 'not-before' date to record the certificate creation time. However, there may be situations where a certificate is issued with a future not-before date.

**Proposition 15** *If required, add an optional* created *field to record the exact creation time of a certificate.*

However, this is not required for transfer certificates, since the 'not-before' date in a transfer certificate's validity field is always its creation time.

### 4.2.5 Data Structures Encoding

The SPKI working group selected the S-Expressions [38] as the standard representation for storing and transmitting of SPKI certificates and related data structures.

Convinced by Carl Ellison's argument against the ASN.1 [39] and failing to find any practically better alternative, I decided to adopt the S-Expressions as well. Appendix B provides the full definitions in BNF for all the data structures discussed in this chapter.

### 4.2.6 Authorization-chain Reduction

The primary responsibilities of a Monitor are authenticating the User, verifying user-supplied credential(s) and either allow or reject Users' requests for accesses or services. In our case, the credentials are in the form of a (chain of) certificate(s). So, the problem here is: How to verify whether a given a chain of certificate(s) yields the privilege requested by the User? Other proposed uses of authorization-chain reduction also include hiding internal chain-of-commands structure and reducing validation delays [24].

Based on the SPKI theory's delegation chain reduction algorithm [12], I would propose the following authorization-chain reduction algorithm:

1. Individual certificates are first verified by verifying the signature, syntax, certificate version and possibly creation date.

2. Certificates are then mapped to intermediate forms, called "*tuples*" here. Authorization certificates and transfer certificates are mapped to "*5-tuples*", whilst name certificates are mapped to "*4-tuples*" (defined below).

3. Where applicable, on-line validation is performed and the result is intersected with the tuple's original capability/role and validity fields.

4. If required, other policy-conformant tests are then performed.

5. (CBAC only) Where applicable, names used in the subjects of capability 5-tuples are reduced to principals using the *Name Reduction Rules* (defined below), based on the name definitions available from accompanying name 4-tuples.

6. 5-tuples are then reduced through the following *Round-trip Reduction* algorithm:

   (a) Applying the *Non-threshold 5-tuples Reduction Rules* (defined below) in a top-down manner, reduce non-threshold 5-tuples, skipping any right-handed open-threshold 5-tuples; if the result still contains more than one 5-tuple then

   (b) Applying the *Threshold 5-tuples Reduction Rules* (defined below) in a bottom-up manner, reduce each group of open-threshold 5-tuples and reduce each resulting 5-tuple with the upper-layer 5-tuple; proceed all the way back to the top to yield the final single 5-tuple.

**5-tuples and 4-tuple Defined**

A Capability Certificate is mapped into a 5-tuple of the form
$\langle \mathbf{I}, [(\mathbf{k\ SID})]\ \mathbf{S}, \mathbf{E}, \mathbf{A}, \mathbf{V} \rangle$, where:

- $I$ is the Issuer's principal (i.e. $K_I$),
- if $(k\ SID)$ is present then $S$ is a threshold subject, where $k$ is the threshold value and $SID$ is the threshold scheme-id,
- $S$ is either the Subject's principal (i.e. $K_S$) or, if indirection through name is used, a local name (i.e. '$K_I\ name_I$' or '$K_S\ name_S$'),
- $E$ is the Empowerment Type and Propagation Control (i.e. $(grant, no-propagation/delegatable/transferable)$ or $(delegate, no-propagation/delegatable)$),
- $A$ is the capability, represented by an SPKI Authorization Tag, and

- $V$ is the Validity conditions field in which:

    1. if not-before date is missing, set it to -$\infty$.
    2. if not-after date is missing, set it to $\infty$.
    3. if maybe-after date is missing, then it would be ignored.

A Role Certificate is mapped into a 5-tuple of the form $\langle \mathbf{I}, [(\mathbf{k\ SID})]\ \mathbf{S}, \mathbf{E}, \mathbf{R}, \mathbf{V} \rangle$, where:

- $I$ is the Issuer's principal (i.e. $K_I$),
- if $(k\ SID)$ is present then $S$ is a threshold subject, where $k$ is the threshold value and $SID$ is the threshold scheme-id,
- $S$ is either the Subject's principal (i.e. $K_S$) or, if re-delegation is permitted, a Subject's local name (i.e. '$K_S\ name_S$'),
- $E$ is the Empowerment Type and Propagation Control,
- $R$ is the Role, in the form of an Issuer's local name (i.e. '$K_I\ name_I$'), and
- $V$ is the Validity conditions.

A Transfer Certificate is mapped into a 5-tuple of the form $\langle \mathbf{I}, \mathbf{S}, (\mathbf{transfer}), (*), \mathbf{V} \rangle$, where:

- $I$ is the Issuer's principal (i.e. $K_I$),
- $S$ is the Subject's principal (i.e. $K_S$),
- $(transfer)$ indicates a privilege transfer (see $EIntersect()$ below),
- $(*)$ indicates a capability/role wild-card (see *5-tuple Reduction Rules* below), and
- $V$ is the Validity conditions.

A Name Certificate is mapped into a 4-tuple of the form $\langle \mathbf{I}, \mathbf{S}, \mathbf{N}, \mathbf{V} \rangle$, where

- $I$ is the Issuer's principal (i.e. $K_I$),
- $N$ is an Issuer's local name (i.e. '$K_I$ $name_{I_1}$'),
- $S$ is either the Subject's principal (i.e. $K_S$), or another local name (i.e. '$K_I$ $name_{I_2}$' or '$K_S$ $name_S$'), and
- $V$ is the Validity condition(s).

**Name Reduction Rules**

$\langle \mathbf{I_1}, [(\mathbf{k_1}\ \mathbf{SID_1})]\ \mathbf{S_1}, \mathbf{E_1}, \mathbf{A_1}, \mathbf{V_1} \rangle + \langle \mathbf{I_2}, \mathbf{S_2}, \mathbf{N_2}, \mathbf{V_2} \rangle$

yields

$\langle \mathbf{I_1}, [(\mathbf{k_1}\ \mathbf{SID_1})]\ \mathbf{S_2}, \mathbf{E_1}, \mathbf{A_1}, \mathbf{VIntersect(V_1, V_2)} \rangle$

where ($k_1$ $SID_1$) may or may not be present and provided:

1. $S_1$ is a local name (i.e. $K_{S_1}$ $name_{S_1}$)
2. $I_2 == K_{S_1}$,
3. $N_2 == S_1$, and
4. $VIntersect(V_1, V_2)$ succeeds.

**Non-threshold 5-tuples Reduction Rules**

**Capability 5-tuples**

$\langle \mathbf{I_1}, [(\mathbf{k_1}\ \mathbf{SID_1})]\ \mathbf{S_1}, \mathbf{E_1}, \mathbf{A_1}, \mathbf{V_1} \rangle + \langle \mathbf{I_2}, \mathbf{S_2}, \mathbf{E_2}, \mathbf{A_2}, \mathbf{V_2} \rangle$

yield

$\langle \mathbf{I_1}, [(\mathbf{k_1}\ \mathbf{SID_1})]\ \mathbf{S_2}, \mathbf{EIntersect(E_1, E_2)}, \mathbf{AIntersect(A_1, A_2)},$
$\mathbf{VIntersect(V_1, V_2)} \rangle$

where ($k_1$ $SID_1$) may or may not be present and provided:

1. $S_2$ is not a threshold subject,
2. both $S_1$ and $S_2$ are principals,

3. $I_2 == S_1$,

4. $EIntersect(E_1, E_2)$ succeeds,

5. $AIntersect(A_1, A_2)$ returns a non-empty tag, and

6. $VIntersect(V_1, V_2)$ succeeds.

**Role 5-tuples**

$\langle \mathbf{I_1}, [(\mathbf{k_1\ SID_1})]\ \mathbf{S_1}, \mathbf{E_1}, \mathbf{R_1}, \mathbf{V_1} \rangle + \langle \mathbf{I_2}, \mathbf{S_2}, \mathbf{E_2}, \mathbf{R_2}, \mathbf{V_2} \rangle$

yield

$\langle \mathbf{I_1}, [(\mathbf{k_1\ SID_1})]\ \mathbf{S_2}, \mathbf{EIntersect(E_1, E_2)}, \mathbf{R_1}, \mathbf{VIntersect(V_1, V_2)} \rangle$

where $(k_1\ SID_1)$ may or may not be present and provided:

1. $S_2$ is not a threshold subject.

2. If the second tuple is a delegation:

   (a) $S_1$ is a local name (i.e. '$K_{S_1}\ name_{S_1}$'),

   (b) $I_2 == K_{S_1}$, and

   (c) $R_2 == S_1$;

   or if the second tuple is a transfer: $I_2 == S_1$,

3. $EIntersect(E_1, E_2)$ succeeds, and

4. $VIntersect(V_1, V_2)$ succeeds.

**EIntersect($\mathbf{E_1}, \mathbf{E_2}$)**

yields one of the following:

- $E_1$, if $E_1 == (grant, transferable)$ and $E_2 == (transfer)$, or
- $(E_1.type, E_2.propagation)$, if $E_1 == (grant/delegate, delegatable)$ and $E_2 == (delegate, no-propagation/delegatable)$, or

- fails otherwise.

**AIntersect($\mathbf{A_1}$, $\mathbf{A_2}$)**

yields one of the following:

- $(*)$, if $A_1 == A_2 == (*)$, or
- $A_1$, if $A_2 == (*)$, or
- $A_2$, if $A_1 == (*)$, or
- (), i.e. an empty tag, if either $A_1 == ()$, or $A_2 == ()$, or
- Some $A_3$, depending on application-defined capability-intersection semantics, or
- (), if the intersection fails.

**VIntersect($\mathbf{V_1}$, $\mathbf{V_2}$)**

yields one of the following:

- (), i.e. always-valid, if $V_1 == V_2 == ()$, or
- $V_1$, if $V_2 == ()$, or
- $V_2$, if $V_1 == ()$, or
- Some $V_3$ where
  - not-before$_3$ = MAX(not-before$_1$,not-before$_2$),
  - maybe-after$_3$ = MIN(maybe-after$_1$,maybe-after$_2$),
  - not-after$_3$ = MIN(not-after$_1$,not-after$_2$), and
  - nonce$_3$ = NUnion(nonce$_1$,nonce$_2$)

  provided:

  1. not-before$_1$ < maybe-after$_1$ $\leq$ not-after$_1$,
  2. not-before$_2$ < maybe-after$_2$ $\leq$ not-after$_2$, and

3. not-before$_3$ < maybe-after$_3$ $\leq$ not-after$_3$

- or fails otherwise.

**NUnion(nonce$_1$, nonce$_2$)**

yields one of the following:

- nonce$_1$, if only nonce$_1$ is present, or
- nonce$_2$, if only nonce$_2$ is present, or
- {nonce$_1$,nonce$_2$}, if both nonce$_1$ and nonce$_2$ are present.

**Open-threshold 5-tuples Reduction Rules**

**Open-threshold Capability 5-tuples**

$\langle \mathbf{I_1}, (\mathbf{k_1\ SID_1})\ \mathbf{S_1}, \mathbf{E_1}, \mathbf{A_1}, \mathbf{V_1} \rangle + \langle \mathbf{I_2}, (\mathbf{k_2\ SID_2})\ \mathbf{S_2}, \mathbf{E_2}, \mathbf{A_2}, \mathbf{V_2} \rangle$

yield

$\langle \mathbf{I_1}, \mathbf{SUnion}((\mathbf{k_1\ SID_1})\ \mathbf{S_1}, (\mathbf{k_2\ SID_2})\ \mathbf{S_2}), \mathbf{EUnion}(\mathbf{E_1}, \mathbf{E_2}), \mathbf{AIntersect}(\mathbf{A_1}, \mathbf{A_2}),$
$\mathbf{VIntersect}(\mathbf{V_1}, \mathbf{V_2}) \rangle$

provided:

1. $I_2 == I_1$,
2. $k_2 == k_1$,
3. $SID_2 == SID_1$,
4. $EUnion(E_1, E_2)$ succeeds,
5. $AIntersect(A_1, A_2)$ returns a non-empty tag, and
6. $VIntersect(V_1, V_2)$ succeeds.

**Open-threshold Role 5-tuples**

$\langle \mathbf{I_1}, (\ \mathbf{k_1\ SID_1})\ \mathbf{S_1}, \mathbf{E_1}, \mathbf{R_1}, \mathbf{V_1} \rangle + \langle \mathbf{I_2}, (\ \mathbf{k_2\ SID_2})\ \mathbf{S_2}, \mathbf{E_2}, \mathbf{R_2}, \mathbf{V_2} \rangle$

yield

$\langle \mathbf{I_1}, \mathbf{SUnion}((\mathbf{k_1\ SID_1})\ \mathbf{S_1}, (\mathbf{k_2\ SID_2})\ \mathbf{S_2}), \mathbf{EUnion}(\mathbf{E_1}, \mathbf{E_2}), \mathbf{R_1}, \mathbf{VIntersect}(\mathbf{V_1}, \mathbf{V_2}) \rangle$

provided:

1. $I_2 == I_1$,

2. $EUnion(E_1, E_2)$ succeeds,

3. $R_2 == R_1$, and

4. $VIntersect(V_1, V_2)$ succeeds.

$\mathbf{SUnion}((\mathbf{k_1\ SID_1})\ \mathbf{S_1}, (\mathbf{k_2\ SID_2})\ \mathbf{S_2})$

yields

$(\mathbf{k_1\ SID_1})\ \{\mathbf{S_1}, \mathbf{S_2}\}$

provided:

1. $k_2 == k_1$,

2. $SID_2 == SID_1$,

$\mathbf{EUnion}(\mathbf{E_1}, \mathbf{E_2})$

yields one of the following:

- $E_1$, if $E_2 == E_1$, or
- $(E_1.type, no-propagation)$, if $E_2.type == E_1.type$ and $E_2.propagation \neq E_1.propagation$, or
- fails otherwise.

### Threshold Elimination Rules

Each time a group of open-threshold 5-tuples with the same scheme-id are successfully reduced into a single 5-tuples with the subject field of the form $(k\ SID)\ \{S_1, ..., S_m\}$, the following procedure is applied:

- If $m < k$ then fails, otherwise
- remove the threshold value and scheme-id (i.e. $(k\ SID)$) from the tuple,
- expand any internal set, i.e. $\{S_1, ..., \{S_a, ..., S_x\}, ..., S_m\}$, into the outer set, i.e. $\{S_1, ..., S_a, ..., S_x, ..., S_m\}$, and then
- eliminate duplications of principals in $\{S_1, ..., S_m\}$ to get $\{S_1, ..., S_n\}$, where $n \geq 1$.

If the resulting set contains only a single principal then it means that the capability/role has propagated—be it through name mapping, delegation, transfer, or some combination thereof—to that particular principal from at least $k$ principals of the original threshold subject. Otherwise, it means that the request must be co-signed by all the principals listed in the subjects set in order to be accepted by the Monitor.

The resulting 5-tuple, i.e. $\langle I, \{S_1, ..., S_n\}, E, A, V\rangle$ for CBAC or $\langle I, \{S_1, ..., S_n\}, E, R, V\rangle$ for RBAC, can then be used as the right-handed 5-tuple to reduce the upper-level 5-tuple using the regular 5-tuple reduction rules by treating the whole set $\{S_1, ..., S_n\}$ as a single subject $S$.

Three examples that demonstrate the *Authorization-chain Reduction Algorithm* are given in Appendix C.

### 4.2.7 Authorization-chain Discovery

While the Monitor's job is to decide whether to authorize or reject each User's request, it is the User who has to come up with the right (chain of) certificate(s) and provide it/them to the Monitor in the right order so as to convince the Monitor to authorize the request. So, the problem here is: Given a pool of certificates, how to construct a chain of certificates that yields the capability/role of interest?

Many have studied the problem and proposed a variety of solutions. The two most current of which are Clarke *et al.*'s Delegation Chain Discovery Algorithm for

SPKI/SDSI [40] and Ninghui Li's logic program for SPKI/SDSI Linked Local Name [26]. Clarke *et al.* also did a run-time analysis on their algorithm and suggested the run-time be bound, at worst, by $O(n^3l)$ for a set of $n$ certificates where $l$ is the length of the longest name in any subject. Li did not provide any run-time analysis or estimation but suggested that their table-based evaluation is more efficient than bottom-up evaluation used by Clarke *et al.*'s algorithm.

On the other hand, as the SPKI theory pointed out, instead of executing a complex authorization-chain discovery algorithm on a (large) pool of certificates, it'd be much more efficient to execute a simple authorization-chain reduction algorithm, such as the one just presented above, against every (chain of) certificate(s) upon receipt and store the resulting 5-tuple and a reference to the input certificate(s) in a (small) look-up cache. The discovery algorithm then becomes a simple search-and-match through a look-up cache, with the run-time bound by $O(m)$ for a set of $m$ authorization chains. Performance could be improved further by maintaining a separate look-up cache for each unique root-Issuer.

### 4.2.8 Ensuring Accountability through Identity and Liability Escrow

So far, we have discussed several measures to protect User's privacy, but have not yet addressed how to balance such protection with the requirement for accountability as mandated by the Design Principles in Chapter 1. To be able to hold a User accountable for its action, a mechanism that enables legitimate investigators to link suspicious entry/entries in the Monitor's audit log back to the real-world entity/entities that caused those entries to be created is needed. Since in this framework the only identifiers in the Monitor's audit log are User's principals, the mechanism must include a mapping between those principals and their corresponding real-world identities.

The SPKI theory mentioned briefly about keeping this *principal* $\mapsto$ *name* mappings as an off-line database maintained by a trusted third party service provider. A few examples [41] of a *"Locator Certificate"*, i.e. a promise to track down the Key-holder for a fees, and an *"Insurance Certificate"*, i.e. a promise to take on a certain amount of financial liability under certain condition were also presented.

Later on, Aura and Ellison [24] addressed the Locator Certificate and proposed the use of a service provider called the *"Escrow Agent"*. An Escrow Agent would maintain a database of the *principal* $\mapsto$ *name* mappings and issue an *"Escrow Certificate"* for each registered principal as proof that the Key-holder's identity has been escrowed. Each Escrow Certificate would also include the conditions for revoking the anonymity, e.g. under court order or for a fees. The User would then supply appropriate Escrow Certificate(s) along with the authorization chain to the Monitor during authentication. They also proposed that the Issuer should be responsible for specifying, as part of the authorization tag or the validity field, the requirements for identity escrow and indicating the trusted Escrow Agent to the Monitor.

However, I argue that there may be situations where:

1. It is the Monitor who requires, and hence demands for proof of, escrow during authentication but the Issuer may not necessarily be aware of, or even care about, such requirements during registration;

2. It is the Issuer who requires, and hence demands for proof of, escrow during registration prior to issuing privileges to Users, but the Monitor may not necessarily be aware of, or even care about, such requirements during authentication; or

3. It is both the Issuer and the Monitor who require escrow and have agreed to cooperate in enforcing such requirements (as envisioned by Aura and Ellison mentioned above).

These conditions mean that we need a more flexible mechanism that can accommodate these different scenarios.

**Proposition 16** *Use an optional* escrow *certificate field, which allows the Issuer to indicate that:*

1. *(escrow field not present) The Issuer did not verify that the subject of the certificate has been escrowed (for identification and/or liability insurance) during registration, and it is up to the Monitor whether to demand for proof of escrow during authentication or not;*

2. *(escrow field present with status* 'to verify'*) The Issuer did not verify that the subject of the certificate has been escrowed during registration, but that the Monitor should demand for proof of escrow—issued either by the Issuer itself or a specified (mutually) trusted Escrow Agent—during authentication, the reference to which could be in form the of a hash of the Escrow Agent's public signature key or an on-line escrow service location; or*

3. *(escrow field present with status* 'verified'*) The Issuer has verified that the subject of the certificate has been escrowed—issued either by the Issuer itself or by a specified (mutually) trusted Escrow Agent—during registration, the reference to which could be in the form of a hash of the Escrow Agent's public signature key, a hash of an Escrow/Insurance Certificate, or an on-line escrow service location.*

The full definition of this field is included in Appendix B. Data structure definitions for the Escrow and Insurance Certificate are, however, left for future works.

## 4.3 The Infrastructure Implementation Guidelines

From the Conceptual Framework proposed above, there are still four outstanding challenges that need to be addressed in order to make the Authorization Management Framework complete and usable. It is also noted that three of those (i.e. Challenge 1, 2 and 4) are concerned with private keys protection and control and one (i.e. Challenge 3) with key management.

One way to satisfy Challenge 1 is to make sure that no entity, not even the key-holder, could ever know what the private key looks like, from the time that key is generated until it is destroyed. However, according to Challenge 2, the key-holder must always have exclusive control over its key. An analogy is a car key that nobody can ever touch or see, and yet only the owner can use it to activate the car's engine without ever touching or seeing it. This implies that the private key must be concealed by some sort of active device that will activate the private key on the key-holder's command without ever revealing the key itself. To be practical and convenient, that device should also be able to manage more than one key-pair, thereby acting like a *key-chain*. On the other hand, each user should, for whatever reason, be able to have and use more than one Key-chain.

**Proposition 17** *A user owns and uses highly-portable, tamper-proof cryptographic device(s) (a.k.a. Key-chain) for key-pair generations and storage and signature and certificate generations.*

Such Key-chain must, at least, be able to:

1. Internally generate and store signature key-pairs and create signatures and certificates from supplied input data. This means that the device must have enough computing power and storage space to carry out those tasks internally with no external assistance, except for, perhaps, power supply;

2. Securely communicate with the key-holder and other entities (i.e. the Issuers, the Monitors and other Users) on behalf of the key-holder. this means that the device must be able to authenticate and distinguish any entity that it communicates with; and

3. Adequately protect itself and its payload against external physical, electrical and logical intrusion and tampering attempts.

Physically, the Key-chain engine may be embedded in a thin rectangular plastic sheet (i.e. smart card), enclosed in a stainless steel case (e.g. the iButton [42]), or any other convenient and robust packages. As long as it complies to the accepted security, functional and inter-operational specifications. As far as current technology goes, the Security Level 3 or, preferably, 4 of the US National Institute of Standards and Technology's FIPS-140 specifications [43, 44] may serve as a basis for the security specifications. Part of the Conceptual Framework proposed above, where it relates to the Key-chain's operations, must be a part of the functional specifications. As for the low-level electrical and logical protocols, parts of the ISO 7816 [45] specifications (e.g. Part 3: Electronic Signals and Transmission Protocols, Part 4: Interindustry Commands for Interchange and Part 8: Security Related Interindustry Commands) could be included in the inter-operational specifications.

Now that we have a computing device, i.e. the Key-chain, that can conceal and has exclusive control over private signature keys, solving Challenge 4 becomes trivial. All that is required is to design and implement the Key-chain's system software so that the private signature key is automatically and permanently deleted from its non-volatile memory immediately after it was used to sign a transfer certificate.

A point to ponder here is that if an authorization management infrastructure is to be built and used within a single and closed corporate environment, then the matter of inter-operation is rather simple. However, as Problem Statement 2 in Chapter 1

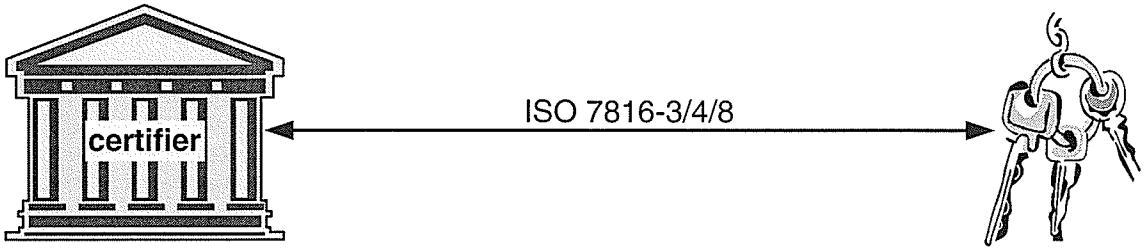


Figure 4.1: The Initialization infrastructure

implies, the solution should be able to utilize the same infrastructure for multiple applications, so that the Users won't have to carry/remember so many credentials as they do today. This means that the infrastructure must be shared amongst many independent Issuers and Monitors which, in turn, makes inter-operability very important and requires cooperation and consensus from all participating entities.

**Proposition 18** *Operate a trusted third-party corporation (a.k.a. the Certifier), preferably owned and/or directed by participating Issuers, Monitors and Users that specifies and conducts compliance tests and certifications of related devices and software, as well as pre-distribution initialization of certified Key-chains.*

It is worth noting that the Certifier only provides technical certification for the infrastructure elements, but not for the participating entities and/or their activities.

Upon passing the compliance testing process, each Key-chain must be initialized by the Certifier (see Figure 4.1) as follows:

1. Reset every single bit of the Key-chain's unoccupied non-volatile memory to the same binary value of either '0' or '1';

2. Store the Certifier's public signature key in the Key-chain's non-volatile memory;

3. Order the Key-chain to generate and store a master signature key-pair in the Key-chain's non-volatile memory;

4. Issue a certificate, signed by the Certifier's private signature key, with the Key-chain's master public signature key as the subject;

5. Store the certificate issued above in the Key-chain's non-volatile memory;

6. Randomly generate and set the Key-chain's initial pass-code.

The Key-chain's system software must be designed and implemented so that i) the Initialization procedure can be performed only once for each Key-chain, and that ii) all data items loaded into the Key-chain during Initialization—except for the initial pass-code—can never be modified for the Key-chain's whole lifetime. The certificate issued by the Certifier authorizes the Key-chain's master key-pair to certify other key-pair(s) generated by the Key-chain as usable key-pairs. The validity period of the certificate should reflect the default lifetime of the Key-chain itself.

Initialized Key-chains can then be distributed to Users through various independent commercial channels, e.g. convenience stores and vending machines, and/or governmental channels, e.g. citizenship department and transportation department. Each User, upon acquiring/receiving a Key-chain, must first Personalize the device as follows:

1. Change to a new pass-code which only the User knows;

2. Order the Key-chain to generate a (specifiable) number of signature key-pairs, certify them with the Key-chain's master key and store them and the associated certificates in a reserved region of the Key-chain's non-volatile memory (a.k.a the Key-pool) for quick use later;

3. Reset every single bit of the Key-chain's unoccupied non-volatile memory to the same binary value of either '0' or '1'.

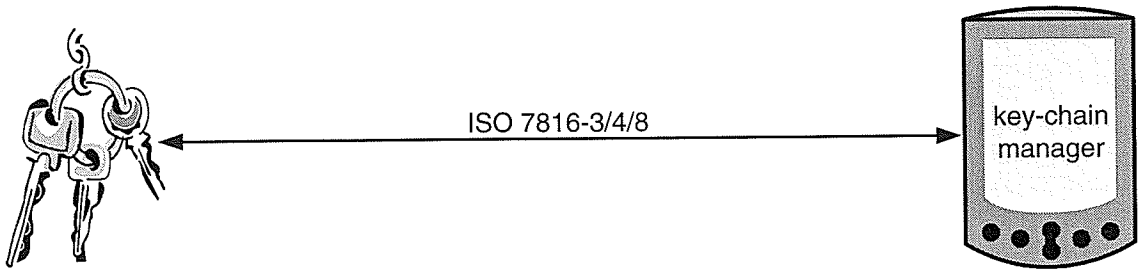


Figure 4.2: The Personalization and Maintenance infrastructure

In contrast to Initialization, Personalization may be performed more than once, provided that the User supplies the correct current pass-code. This allows the User to reset the Key-chain, which was somehow corrupted, back to the Initialized state in-the-field without compromising the integrity of the Key-chain itself or the User's private data.

Once Personalized, the User will need to perform a number of Maintenance tasks from time to time as follows:

1. Occasionally change the pass-code; and

2. Occasionally update the Key-pool with a (specifiable) number of newly generated and certified key-pairs when the Key-pool's reserves drop below a certain (specifiable) threshold.

In most existing implementations of access-control systems that make use of portable cryptographic device, a User would have to rely on either user-interface devices installed at the access/service points (e.g. point-of-sales terminals and banking machines) or general-purpose computers (e.g. PCs and PDAs) in order to interact with the device. Such systems extended the User's Trusted Computing Base (TCB) to include third-party controlled and/or easily compromised and/or potentially hostile environments. However, the Design Principles in Chapter 1 mandate that the system must keep the Users in the decision-making loop at all times. So, we need

to design the infrastructure in such a way that Users can interact autonomously and privately with their Key-chains anytime and anywhere at will [46, 47].

**Proposition 19** *User always uses a portable and personal user-interface device (a.k.a. Key-chain Manager) to interact with its key-chain(s).*

The Key-chain and the Key-chain Manager together comprise a TCB or, in other words, the User's Private Domain. The presence of a private and direct communication channel between the Key-chain and its owner also makes Challenge 3 easy to solve.

The rather simple but complete infrastructure required for Personalization and Maintenance is shown in Figure 4.2. By decoupling the Key-chain Manager from the Key-chain and standardizing on a common interfacing specifications, manufacturers can—actually should—make various models of the Key-chain Managers that offer different types of user-interfacing methods. This may include voice-controlled for the visual or motor impaired and graphical input/display for the vocal/hearing impaired. Also, regular Users might prefer different types of the device. Some (who can afford to) might even want to keep more than one for different purposes. For example, a voice-controlled device in the car for hands-free operations while driving, a rugged and water-proof one for operations in tough terrains and a very thin and small one with touch-screen in the pocket for regular uses.

The Key-chain Manager, like the Key-chain, should go through the Certifier's certification process. But, unlike the Key-chain, it should not need to be initialized by the Certifier. Proof-of-certification may even be in physical form such as a tamper-evident holographic sticker put strategically on the device. A person should be able to switch between different Key-chain Managers at will or even borrow a friend's Key-chain Manager to use with own's Key-chain when needed. For convenience, some operational parameters of the Key-chain Manager could be made configurable by the

User. This may include how long a pass-code should be cached in the Key-chain Manager so that the User won't have to re-enter it too often. Furthermore, the User should be able to assign and refer to each key-pair with an easy-to-remember name that reflect the key-pair's application such as 'intranet key', 'driver's license key' and 'student key'.

At this point, the Users are fully-equipped and ready to register and acquire/receive the various privileges from the Issuer(s) they are affiliated with. However, let's first take a moment and revisit the issue of the difference between Issuers that are actual *authority* vs. mere *trusted third-party* mentioned earlier in Chapter 3.

What we are seeing more and more from *"Third Party"* claiming to be *"Certification Authority"* (CA) are paragraphs of the so-called *fine-prints* in the Service Agreements and/or Certificate Policy. These fine-prints practically disclaim the *"CA"* from any and all liabilities incurred directly or indirectly by/from the use of any certificate issued by the *"CA"*. Yet, these *"CAs"* impose the fee structures for their services in such a way that make it simply impractical for users to have and maintain multiple key-pairs for use with different applications/roles. Suffice it to say that users/customers of these CAs are hardly getting any assurance/protection from these self-appointed *"authorities"* for their transactions/sessions at all. Compare such practice to that of authentic trusted third-parties like banks and credit-card issuers, who actually stand behind their services and accept reasonable liabilities, and we begin to see how bogus these *"CAs"* really are.

The only reason that people are *buying* in to this hoax is because the *"security experts"* and *"consultants"*, often employed directly or indirectly by those CAs or related equipments/software suppliers—using the extremely thick X.509 specifications or its derivatives and other scarecrow as evidences—have been *"educating"* us with misleading *"information"*. These information suggested that i) managing privileges

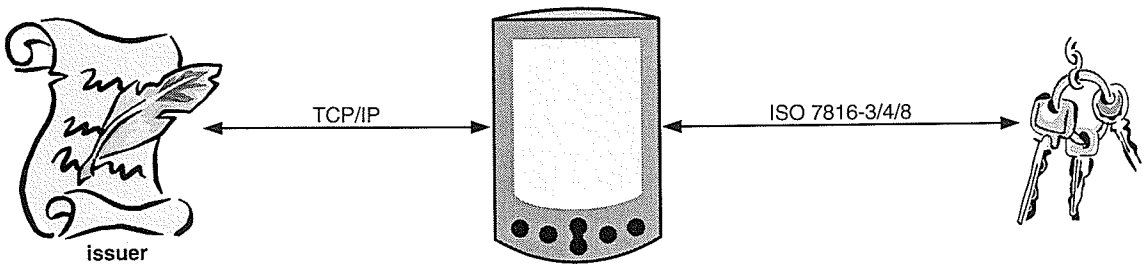


Figure 4.3: The Registration infrastructure

using public-key cryptographic technologies is prohibitively expensive and complicated, and that ii) there is no other viable alternative technologies and/or business models. However, the simplicity of the framework presented so far suggests to the contrary. There is no need to mindlessly rely on third parties for internal authorization management. Unless, it is a well thought-out risk-management decision to delegate (parts of) the responsibilities to *reliable* service provider(s), who are ready to act on behalf of and share any associated liabilities with the hiring entities.

**Proposition 20** *Privilege issuance, suspensions and revocations are handled only by entities responsible for and/or authorized to perform such operations regarding those particular privileges.*

Although the full Registration procedures will differ from one Issuer to another, they must include the following steps:

1. Verify that the particular Key-chain is certified by authenticating the Key-chain's master key-pair against the Certifier-issued certificate;

2. Store the Issuer's public signature key in the Key-chain's non-volatile memory, thereby creating a new Domain within the Key-chain identified by the (hash of the) Issuer's public signature key;

3. Issue authorization/name certificate(s) as needed, using public signature key(s), certified by the Key-chain's master key-pair, taken from the Key-chain's Key-pool as subject(s);

4. Store the certificate(s) issued above in the Key-chain's non-volatile memory under the Issuer's Domain;

5. Generate or update a look-up cache for the particular Issuer on the Key-chain Manager for fast authorization chain discovery later;

6. (optional) Register the User's identity for escrow.

Registration is yet another occasion when the User's Key-chain Manager plays a significant role, as depicted in Figure 4.3. Instead of forcing Users to put themselves at the mercy of the (partially trusted) Issuer, the Users could use their Key-chain Manager to proxy the communication between their Key-chain and the Issuer's systems. This would allow the users to monitor the Registration process as it happens, and be able to authorize or reject the process as they see fit. To make this works properly, the Key-chain's system software must require that all inquiries and commands sent to it must be accompanied by the correct pass-code in order to be accepted and executed. Communications between the Key-chain Manager and external systems should be based on the robust and globally accepted TCP/IP protocol suite. For improved performance, the Key-chain Manager should also cache all certificates issued by/to the Key-chain in its internal memory for fast retrieval when needed.

To protect against brute-force attack on the Key-chain's pass-code, the Key-chain's authentication function could add an increasingly long delay to its response time after receiving an incorrect pass-code. For example, the delay could be set to $3^n - 1$ seconds, where $n$ is the number of bad pass-codes received consecutively, which makes for over an hour of response time after only 8 tries. This idea was in-

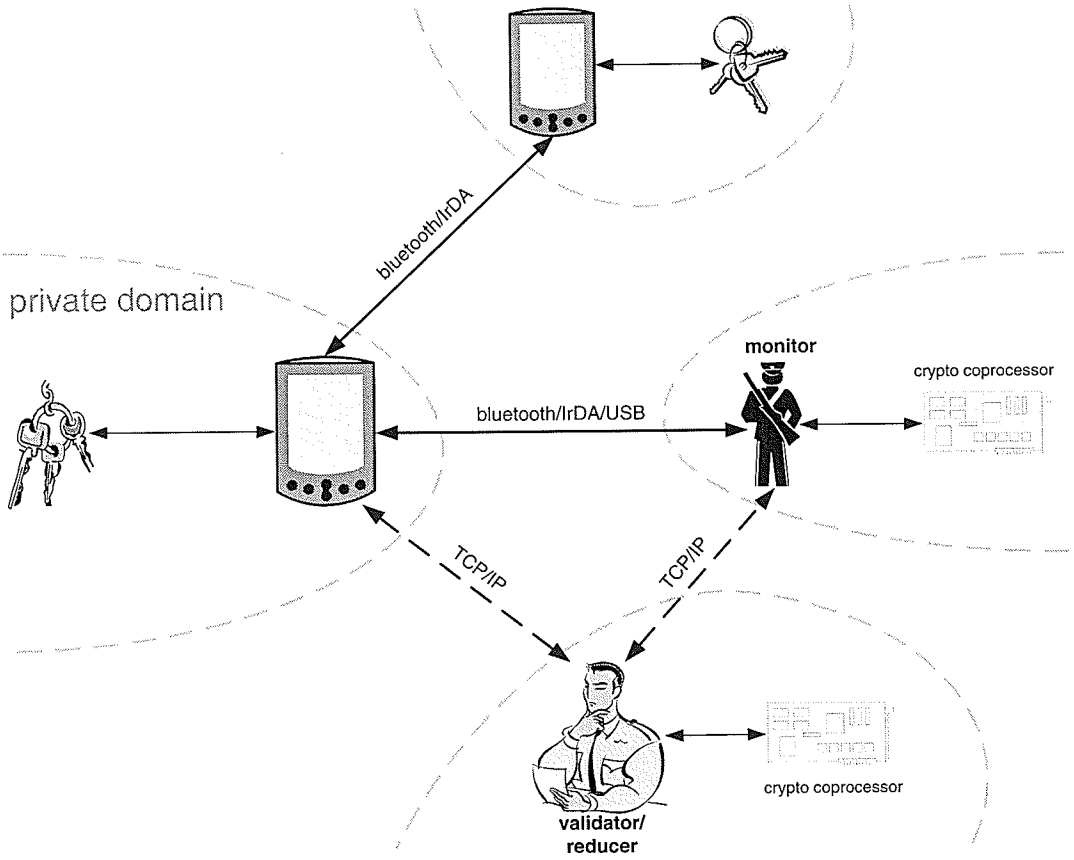


Figure 4.4: The Utilization and Propagation infrastructure

spired by Kelsey & Schneier's work on secure token authentication using slow memory [48]. Another practical and complementary approach is to allow for variable-length hexadecimal or even alphanumeric pass-codes with a (specifiable) minimum length, instead of the fixed-length 4-digit decimal PINs most systems use.

Once registered, the User can then utilize received privileges and, where permitted, propagate its privileges to and receive more privileges from other Users as follows:

1. Authenticate and supply the required authorization chain to the Monitor when requesting for access/service;

2. Empower another User by issuing name/authorization/transfer certificate(s), using public signature key(s) supplied by that User as subject(s) and send the

resulting (chain of) certificate(s) to that User;

3. Supply public signature key(s)—taken from the Key-pool—to another User, receive and store new certificate(s) under the appropriate Domain(s) in the Key-chain and update the look-up cache(s) on the Key-chain Manager;

Here again, the Key-chain Manager is heavily used to always put the User in the decision-making loop, as shown in Figure 4.4. Whether authenticating to the Monitor, interacting with another User, or communicating with the online authorization-chain reduction/validation server, the Key-chain Manager enables the User to always be aware of the transaction's actual details. This effectively allows the User to make an informed decision that would best reflect and protect the User's intent and interest, respectively. As for the monitor and the reduction/validation server, they should be equipped with high-performance secure cryptographic co-processor similar to that proposed by Gutmann in [47].

Depending on the adopted business model, financing for production and distribution of the Key-chains and the Key-chain Managers and operation of other infrastructure elements can be subsidized in parts by the participants. The key idea here is: The more participants the shared infrastructure has, the more economical it will be for everyone involved.

# Chapter 5

# Reference Implementation

In the following sections, I briefly describe a java class library implementation of the Authorization Management Framework (AMF), its limitations and the performance benchmarking results.

## 5.1 The Java Class Library and Demo Program

The reference java class library consists of the *spki* package and the *spki.interfaces* package, plus the *spki.demo* demonstration program package. The class library structure (see Figure 5.1) and code-base are based loosely on and adapted from Per Harald Myrvang's *SPKI* class library [49].

The Authorization Chain Reduction algorithm and related objects are implemented by the *spki.Tuple*, *spki.Name4Tuple*, *spki.Capability5Tuple*, *spki.Role5Tuple*, *spki.CBACChainReducer* and *spki.RBACChainReducer* classes. To the best of my knowledge, this class library is the first publicly-available implementation of the Openthreshold Subject concept.

The *spki.test* and *spki.benchmark* programs provide tools for the class library's functionality demonstration and performance benchmarking (see Section 5.3 for re-

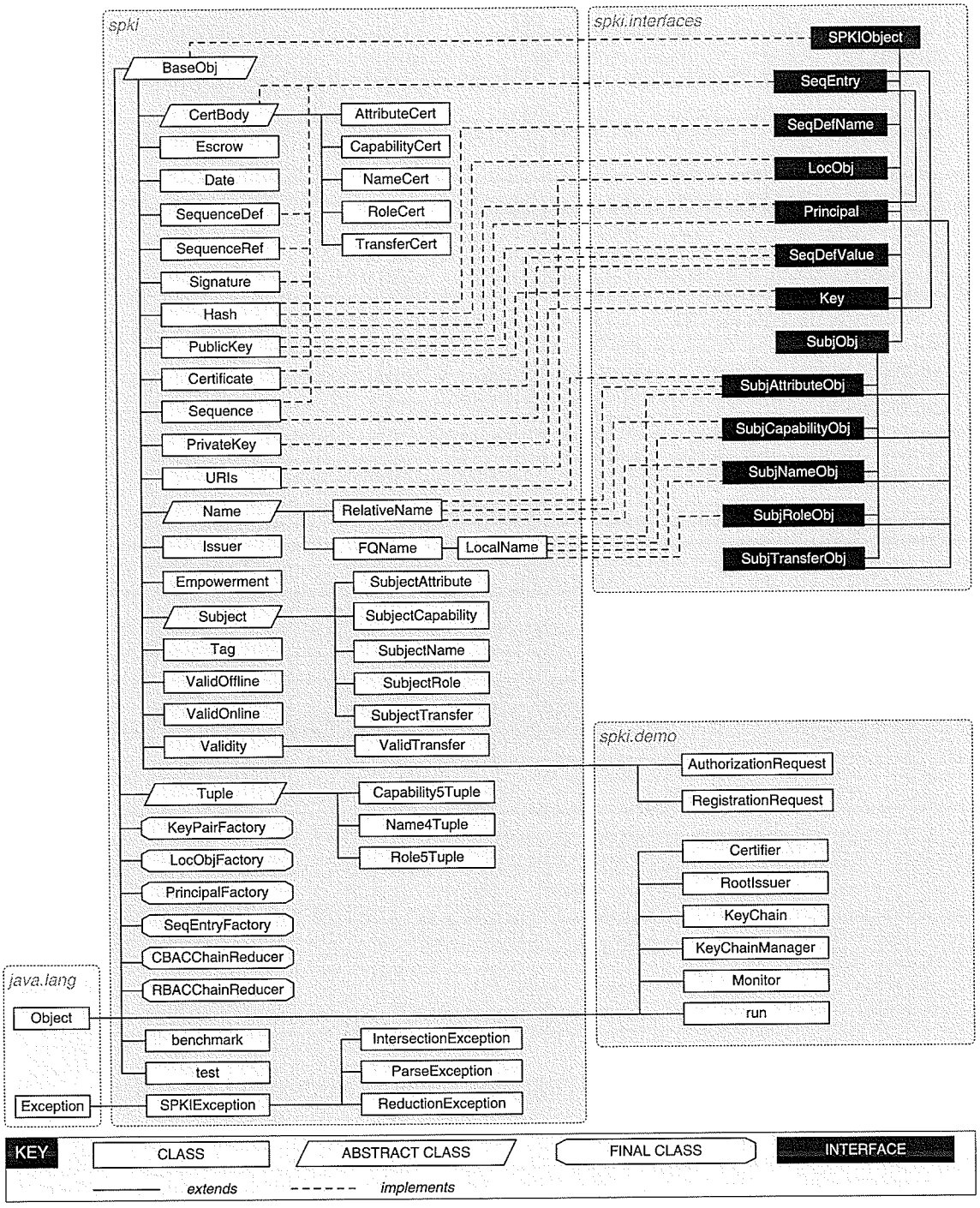


Figure 5.1: The AMF java class library

sults), respectively. In addition, there is also the *spki.demo.run* program that provides a simple simulation of the Infrastructure Implementation Guidelines presented in Chapter 4.

Please refer to Appendix D for information on how to obtain the full API documentation and source code for the class library and programs.

The development platform and third-party software used are as follows:

1. The development workstation was a portable computer with Celeron 400MHz CPU, originally with 64M bytes and later upgraded to 192M bytes of main memory, running Linux kernel version 2.4.10-based operating system.

2. Version 1.2.2 of the Java Development Kit (build JDK-1.2.2_008) was used for compiling the source code, generating the javadoc-based API documentations and running the compiled byte-code.

3. Version 3.2.0 of the Cryptix class library [50] was used for all public-key cryptographic operations.

4. After a little debugging, Myrvang's *sexp* class library [49], was used to represent the various S-Expression primitives defined in Appendix B.

5. After a little debugging, the *HashDB* part of Myrvang's *database* class library [49], was used for non-volatile storage of configuration data, key-pairs and certificates by the *spki.test* and *spki.demo.run* programs.

6. Version 1.0.5 of the GNU *getopt* class library was used for command-line options parsing of the *spki.test*, *spki.benchmark* and *spki.demo.run* programs.

7. Version 1.0.6 of the GNU *regexp* class library was used for regular-expression processing by Myrvang's *sexp* class library.

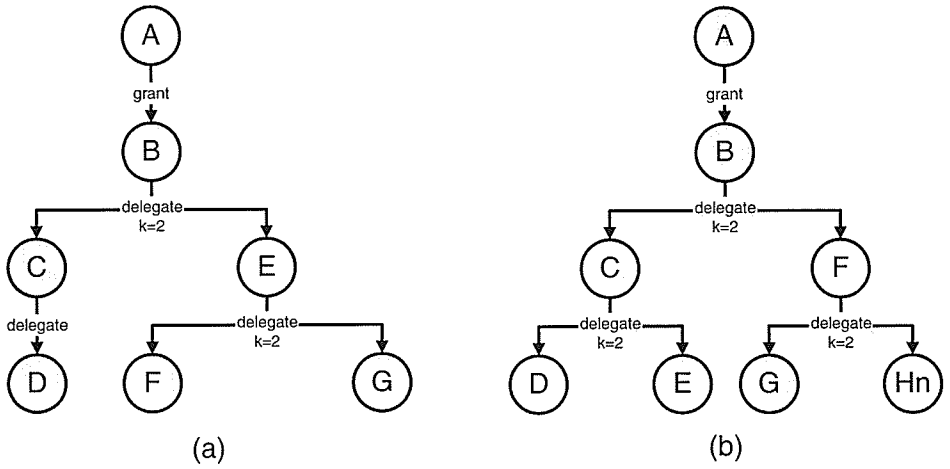


Figure 5.2: Examples of (a) un-balanced and (b) balanced thresholded authorization chains

## 5.2 Limitations and Extensions

Following are the limitations of the class library implementation as compared to what was presented in Chapter 4 and Appendix B:

1. All data structures defined in Appendix B are implemented except for the `<acl-entry>`, `<acl>`, `<reduce-op>`, `<general-op>` and `<op>` structures, which are not essential for the proof-of-concept prototypes.

2. The `<alg-id>` structure is represented by the `java.lang.String` class and can only handles algorithm-name string. Using URI string as algorithm identifier is not supported.

3. The `<sig-params>` structure is represented by the `sexp.Sexp` class, which implements the `<byte-string>` primitive. Signature parameters using the `<s-expr>` structure(s) is not supported.

4. Only the SHA-1 and RSA algorithms are supported for hashing and public-key cryptography, respectively. Addition of other cryptographic algorithms and/or

migration to other cryptographic libraries can be achieved through modifications to the *spki.KeyPairFactory*, *spki.PrivateKey*, *spki.PublicKey*, *spki.Hash* classes and the *spki.BaseObj.extractHashAlgId()*, *spki.BaseObj.spkiToProvider()* and *spki.BaseObj.providerToSPKI()* utility methods. All other classes are algorithm independent and should not require any modification.

5. The *spki.Tuple.NUnion()* method, which implements the $NUnion(nonce_1, nonce_2)$ algorithm, does not support cases where both $nonce_1$ and $nonce_2$ are present but not equal.

6. The *spki.Capability5Tuple.AIntersect()* method, which implements the $AIntersect(A_1, A_2)$ algorithm, only deals with wildcard, empty and equivalent tags. Application-specific capability intersection algorithm may be implemented as an overriding method.

7. The *spki.CBACChainReducer.bottomUpPass()* and *spki.RBACChainReducer.bottomUpPass()* methods, which implement the main loop of the Open-threshold Reduction Rules, can only deal with un-balanced thresholded authorization chains, such as that in Figure 5.2a. Reduction of balanced thresholded authorization chains, such as that in Figure 5.2b, is not supported.

## 5.3 Performance Benchmarking

The the *spki.benchmark* program were executed on the same system as the development platform described above. The difference is that, during the benchmark runs, the system was running in single-user mode with all non-essential peripherals and software disconnected and disabled, respectively.

The following benchmarking tests were performed:

Key-pair Generations

| Key Size (bits) | Time (msec) |
|---|---|
| 128 | 126.68 |
| 256 | 205.26 |
| 384 | 374.74 |
| 512 | 653.83 |
| 640 | 997.84 |
| 768 | 2,010.70 |
| 896 | 2,617.66 |
| 1,024 | 3,447.80 |
| 2,048 | 28,566.20 |

SHA-1 Hashing

| Plaintext Size (bytes) | Time (msec) |
|---|---|
| 128 | 0.754 |
| 256 | 1.401 |
| 384 | 1.829 |
| 512 | 2.264 |
| 640 | 2.693 |
| 768 | 3.127 |
| 896 | 3.561 |
| 1,024 | 3.989 |
| 10,240 | 35.056 |
| 102,400 | 346.020 |

rsa-pkcs1-sha1 Signature Creation

| Plaintext Size (bytes) | Key Size | | | | | | |
|---|---|---|---|---|---|---|---|
| | 384 bits | 512 bits | 640 bits | 768 bits | 896 bits | 1024 bits | 2048 bits |
| 128 | 6.69 | 10.29 | 15.16 | 21.95 | 31.00 | 42.86 | 267.87 |
| 256 | 7.42 | 10.59 | 15.45 | 22.60 | 31.90 | 43.76 | 268.20 |
| 384 | 7.55 | 11.12 | 15.97 | 22.75 | 32.47 | 44.00 | 269.13 |
| 512 | 8.12 | 11.95 | 16.29 | 23.67 | 32.80 | 44.19 | 269.09 |
| 640 | 9.14 | 11.98 | 16.80 | 23.76 | 33.12 | 44.40 | 268.99 |
| 768 | 8.81 | 12.43 | 17.25 | 24.09 | 33.61 | 45.26 | 270.46 |
| 896 | 9.30 | 12.72 | 17.61 | 24.67 | 33.96 | 45.61 | 271.20 |
| 1,024 | 10.07 | 13.48 | 18.44 | 25.05 | 35.01 | 46.11 | 269.32 |
| 10,240 | 41.15 | 44.61 | 49.56 | 56.33 | 65.48 | 77.39 | 301.43 |
| 102,400 | 351.84 | 355.35 | 360.49 | 367.10 | 376.09 | 389.19 | 612.21 |

rsa-pkcs1-sha1 Signature Verification

| Plaintext Size (bytes) | Key Size | | | | | | |
|---|---|---|---|---|---|---|---|
| | 384 bits | 512 bits | 640 bits | 768 bits | 896 bits | 1024 bits | 2048 bits |
| 128 | 3.16 | 3.39 | 3.70 | 4.05 | 4.07 | 4.55 | 10.83 |
| 256 | 3.57 | 3.81 | 4.10 | 4.52 | 4.98 | 5.01 | 11.20 |
| 384 | 4.00 | 4.26 | 4.58 | 4.92 | 4.93 | 5.42 | 11.65 |
| 512 | 4.44 | 4.29 | 4.98 | 5.37 | 5.36 | 6.30 | 11.78 |
| 640 | 4.88 | 5.16 | 5.41 | 5.80 | 5.82 | 6.28 | 12.76 |
| 768 | 5.30 | 5.57 | 5.83 | 5.83 | 6.26 | 6.72 | 13.12 |
| 896 | 5.77 | 5.99 | 6.29 | 6.64 | 6.67 | 7.18 | 13.41 |
| 1,024 | 6.19 | 6.02 | 6.74 | 7.11 | 7.13 | 7.58 | 13.87 |
| 10,240 | 37.31 | 37.55 | 37.42 | 38.19 | 38.20 | 38.70 | 44.89 |
| 102,400 | 348.18 | 348.11 | 348.79 | 349.13 | 349.54 | 349.97 | 355.86 |

Table 5.1: Benchmarking results

1. RSA key-pair generation with key-sizes 128, 256, 384, 512, 640, 768, 896, 1024 and 2048 bits.

2. SHA-1 hashing of pre-generated plaintext with sizes 128, 256, 384, 512, 640, 768, 896, 1024, 10240 and 102400 bytes.

3. rsa-pkcs1-sha1 signing and verification, with key-sizes 384, 512, 640, 768, 896, 1024 and 2,048 bits, of pre-generated plaintext with sizes 128, 256, 384, 512,

640, 768, 896, 1024, 10240 and 102400 bytes.

4. Reduction of CBAC delegation, transfer, name-mapping and thresholded delegation chains with chain-depths 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100.

5. Reduction of RBAC delegation, transfer and thresholded delegation chains with chain-depths 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90 and 100.

Following are the notables characteristics of the results shown in Table 5.1, 5.2 and plotted in Figure 5.3.

1. *Key-pair Generation*, *Plaintext Hashing*, *Signature Creation* and *Signature Verification* processes make extensive use of the underlying cryptographic libraries, as explained earlier in Section 5.1. Thus a move to another cryptographic class library may alter the results significantly.

2. *Key-pair Generation* times increase with only a slight acceleration from 128 to 640 bits key-sizes before doubling at 768 bits. More notable, however, is the sharp jump of almost an order-of-magnitude from 1,024 to 2,048 bits.

3. *Plaintext Hashing* time increments are largely linear throughout the whole range of plaintext sizes.

4. *Signature Creation* times for all key sizes, with an exception for key-size 2,048 bits, converge around 350 to 390 milliseconds when the plaintext size reach 102,400 bytes.

5. *Signature Verification* times for all key sizes, with no exception, converge around 350 milliseconds with the plaintext size reaches 102,400.

6. *Authorization Chain Reduction* times for CBAC and RBAC delegation chains and transfer chains at depth 1 are all in the order of 0.6 microseconds.

7. *Authorization Chain Reduction* times differences for CBAC delegation chains, transfer chains and RBAC transfer chains at every depth are, at most, within just a few milliseconds.

8. On the other hand, *Authorization Chain Reduction* times for RBAC delegation chains at depths 5 to 100 are all roughly 4 times lower than those for equivalent CBAC delegation chains. However, all attempts to track down the cause for such differences in the class library implementation did not turn up anything substantial.

Since optimum performance is not part of the goals for the development of this class library, all benchmarking results are presented for reference purposes only. Please also note that in an attempt to show all data points clearly, all charts in Figure 5.3 are plotted using logarithmic scale on the Y-axis.

Please refer to Appendix D for information on how to obtain the benchmarking environment's run-time status (i.e. outputs of the *top*, *ps*, *uname* and *swapon* commands) and raw outputs of the benchmark runs.

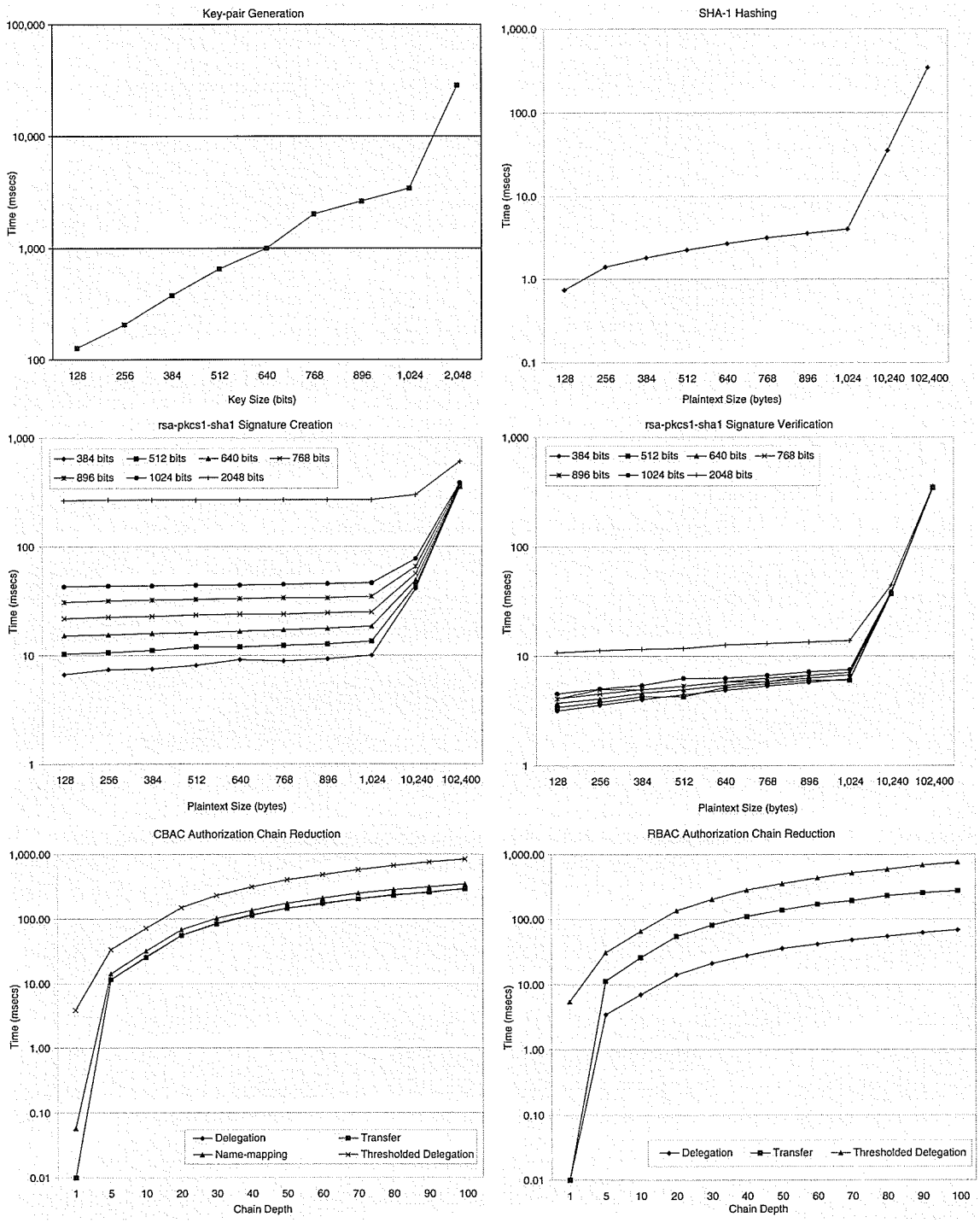


Figure 5.3: Benchmarking results

**CBAC Authorization Chain Reduction**

| Chain Depth | Chain Type | | | |
|---|---|---|---|---|
| | Delegation | Thresholded Delegation | Transfer | Name-mapping |
| 1 | 0.0000 | 3.9064 | 0.0000 | 0.0570 |
| 5 | 11.6404 | 33.4906 | 11.4912 | 14.0642 |
| 10 | 26.1920 | 70.6770 | 25.4490 | 31.9150 |
| 20 | 55.6750 | 148.8320 | 55.2830 | 67.6970 |
| 30 | 85.9710 | 227.1390 | 82.9130 | 102.4980 |
| 40 | 115.6420 | 309.2470 | 113.6340 | 136.1130 |
| 50 | 144.8830 | 398.2470 | 145.2020 | 174.5790 |
| 60 | 171.2670 | 484.4680 | 174.2830 | 206.7120 |
| 70 | 201.8610 | 568.6730 | 201.9230 | 244.9340 |
| 80 | 233.7240 | 665.4370 | 231.5090 | 279.6100 |
| 90 | 257.0050 | 755.1190 | 260.4490 | 311.0960 |
| 100 | 293.6380 | 842.1340 | 289.2500 | 342.1950 |

**RBAC Authorization Chain Reduction**

| Chain Depth | Chain Type | | |
|---|---|---|---|
| | Delegation | Thresholded Delegation | Transfer |
| 1 | 0.0000 | 5.3986 | 0.0000 |
| 5 | 3.4116 | 31.2690 | 11.4170 |
| 10 | 7.0680 | 65.3840 | 25.7700 |
| 20 | 14.3250 | 137.8170 | 55.2970 |
| 30 | 21.1060 | 205.1030 | 82.3210 |
| 40 | 28.2080 | 280.9280 | 111.3820 |
| 50 | 35.9030 | 357.3110 | 140.6560 |
| 60 | 42.3360 | 437.2710 | 170.6780 |
| 70 | 48.6910 | 516.9830 | 195.8740 |
| 80 | 55.5140 | 593.3920 | 229.1810 |
| 90 | 63.0680 | 680.6950 | 254.8970 |
| 100 | 68.9020 | 764.9700 | 280.0760 |

Table 5.2: Benchmarking results (cont.)

# Chapter 6

# Conclusions and Future Works

This concluding chapter discusses the contributions of this work as well as propose directions for future research.

## 6.1 Contributions

It has been shown in this thesis that public-key cryptography can be practically and effectively used as a tool in an electronic authorization management solution that is geared towards balancing the needs for privacy protection and accountability preservation. Also, it has been shown how such a solution may be deployed in a manner that could reduce the number of credentials and/or secrets a user has to carry and/or remember in order to utilize the authorization management system.

Specific contributions of this thesis include:

1. A real-world modeling of the authorization management problem;

2. A novel and simple Authorization Management Framework that applies public-key cryptography to solving the problem;

3. A suite of practical guidelines for implementing an infrastructure to support the framework; and

4. A reference implementation of the proposed framework.

## 6.2 Future Work

Following are the selected outstanding issues that are potential candidate for future works. The list is by no means exhaustive but is intended to offer pointers for some interesting directions for future research.

### 6.2.1 Long-lived Signatures

In the presented framework, I only discussed sessional and transactional authorization management and intentionally left out applications of digital signatures on long-lived objects and documents such as electronic mails and legal contracts. This is because such applications require additional non-trivial infrastructure to (re)validate and (re)verify the authenticity of signed objects beyond the useful lifetime of the signing private signature key itself. These infrastructures may include, but are not limited to, time-stamping, notary, and archiving services usually performed by neutral trusted third parties. The operational and auditory framework for these service providers are non-trivial and often charged with social, economical and political issues.

### 6.2.2 Attributes Certification

I also did not fully address an authorization management system where the Issuer (a.k.a. *Certification Authority* or *Trusted Third Party*) certifies User's certain attributes (e.g. date-of-birth, marital status), without having to know or care for what

purpose and by whom the certified attributes will be interpreted and used; whilst the Monitor independently uses the certified attributes in its authorization decision. An example of this is how bars or merchants verify customer's age using the date-of-birth printed on driver's license or some form of identification before selling alcoholic beverages or cigarettes.

Note that attribute certification does not imply empowerment (i.e. authorization), therefore the Monitors in such system are on their own as far as liability is concerned, unless warranted otherwise by the Issuer. Another notable issue with Attribute Certification is that it may reveal too much information about the User to the Monitor, thereby impeding the User's privacy. So, care must be taken not to unnecessarily put too many attributes on the same certificate.

On the other hand, by extension, designing a certificate structure for Attribute-based Access Control (ABAC) is rather trivial when compared to those for RBAC and CBAC, since propagation is never allowed. So, I have also included a sample of the full definitions for Attribute Certificate in Appendix B.

### 6.2.3 Encryption Key-pairs Management

By extension, a User should be able to autonomously generate and empower an encryption key-pair by delegating privileges from the signature key-pair used in the session/transaction. Backup of sessional/transactional private decryption keys is irrelevant and, therefore, escrow of such private encryption keys must be strictly prohibited. Note that, unlike the signature key-pair, an encryption key-pair cannot propagate received privileges further since it cannot *"speak for"* the key-holder through digital signature and issue certificates.

On the other hand, encryption key-pairs used for long-lived objects (e.g. emails and legal documents) require special treatments. This is because a loss of a decryption

private key for such object means that the object itself is also lost forever. Such a loss may cause significant damage to the key-holder and, in some cases, even third-parties (e.g. key-holder's employer). So, backup and escrow of private decryption keys for long-lived objects may be optional and must be subject to the mutual consent between the key-holder and the relying entity.

An interesting area for future work is to use public encryption key as the subject of authorization/name certificates.

### 6.2.4 Biometrics Authentication

Another authentication method that is gaining popularity is biometrics. Biometrics use certain unique physical (e.g. fingerprint, voice-print and facial structure) or behavioral (e.g. keyboard typing and ink-signature signing) patterns of a user for authentication.

Like encryption key-pair, however, biometric template cannot *"speak for"* its owner, and so is not considered a principal. However, attribute certification, perhaps operated by a trusted third party, can be used to bind a person's biometric template to the person's principal for use in the authentication process. Another way is to delegate privileges directly to the biometric template through an authorization certificate.

Another challenging area for future work is to study the trade-offs between these and other potential solutions. It is also charged with social and political issues.

# Appendix A

# Traditional Definitions of Terms

SOURCE: Oxford Advanced Learner's Dictionary, 6th Edition.

**accountable** (adj) responsible for your decisions or actions and expected to explain them when you are asked.

**agent** (n) a person whose job is to act for, or manage the affairs of, other people in business, politics, etc.

**anonymity** (n) the state of remaining unknown to most other people.

**attribute** (n) a quality or feature of somebody/something.

**audit** (n) an official examination of business and financial records to see that they are true and correct.

**authenticate** (v) To prove that something is genuine, real or true.

**authority** (n) the people or an organization who have the power to make decisions or who have a particular area of responsibility in a country or region.

**authorization** (n) official permission or power to do something; the act of giving permission.

**confidentiality** (n) a situation in which you expect somebody to keep information secret.

**credential** (n) documents such as letters that prove that you are who you claim to be, and can therefore be trusted.

**data** (n) information that is stored by a computer.

**delegate** (v) to give part of your work, power or authority to somebody in a lower position than you.

**entity** (n) something that exists separately from other things and has its own identity.

**grant** (v) to agree to give somebody what they ask for, especially formal or legal permission to do something.

**identification** (n) the process of showing, proving or recognizing who or what somebody/something is.

**identity** (n) who or what somebody/something is; the characteristics, feelings or beliefs that distinguish people from others.

**information** (n) facts or details about somebody/something.

**integrity** (n) the quality of being honest and having strong moral principles.

**permission** (n) the act of allowing somebody to do something, especially when this is done by somebody in a position of authority; an official written statement allowing somebody to do something.

**principal** (n) a person that you are representing, especially in business or law: The shareholders are principals and the managers are agents.

**privacy** (n) the state of being alone and not watched or disturbed by other people; the state of being free from the attention of the public.

**privilege** (n) a special right or advantage that a particular person or group of people has.

**pseudonym** (n) a name used by somebody, especially a writer, instead of their real name.

**risk** (n) the possibility of something bad happening at some time in the future; a situation that could be dangerous or have a bad result.

**role** (n) the function or position that somebody has or is expected to have in an organization, in society or in a relationship.

**security** (n) the activities involved in protecting a country, building or person against attack, danger, etc.

**session** (n) a period of time that is spent doing a particular activity.

**sign** (v) to write your name on a document, letter, etc. to show that you have written it, that you agree with what it says, or that it is genuine.

**third party** (n) a person who is involved in a situation in addition to the two main people involved.

**transaction** (n) a piece of business that is done between people, especially an act of buying or selling.

**transfer** (v) to officially arrange for something to belong to somebody else or for somebody else to control something.

**trust** (n) the belief that somebody/something is good, sincere, honest, etc. and will not try to harm or deceive you.

**validate** (v) to prove that something is true.

**verify** (v) to check that something is true or accurate.

# Appendix B

# Data Structures and Definitions

NOTE: Adapted from the draft SPKI certificate structure specification [36].

## Primitives

```
<nzddigit>:: "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
<ddigit>:: "0" | <nzddigit> ;
<nzdecimal>:: <nzddigit> <ddigit>* ;
<decimal>:: <nzdecimal> | "0" ;
<length>:: <nzdecimal> ;
<bytes>:: <length> ":" {binary byte string of that length} ;
<mime-type>:: {ASCII-encoded <bytes> representing a MIME-type string} ;
<display-hint>:: "[" <mime-type> "]" ;
<value>:: <bytes> ;
<byte-string>:: <display-hint>? <value> ;
<integer>:: {<byte-string> representing an integer in network byte-order} ;
<s-type>:: <byte-string> ;
<s-part>:: <byte-string> | <s-expr> ;
<s-expr>:: "(" <s-type> <s-part>* ")" ;
```

## Location, Key, Hash, Principal, Signature, and Names

```
<uri>:: <byte-string> ;
<uris>:: "(" "uri" <uri>+ ")" ;
<alg-id>:: <byte-string> | <uri> ;
<hash-alg-id>:: <alg-id> ;
<hash-value>:: <byte-string> ;
<hash>:: "(" "hash" <hash-alg-id> <hash-value> <uris>? ")" ;
<loc-obj>:: <uris> | <hash> | <s-expr> ;
<pub-sig-alg-id>:: <alg-id> ;
<pub-key>:: "(" "public-key" "(" <pub-sig-alg-id> <s-expr>+ ")" ")" ;
<hash-of-key>:: <hash> ;
<principal>:: <pub-key> | <hash-of-key> ;
<hash-of-obj>:: <hash> ;
<signer>:: <principal> ;
<sig-params>:: <byte-string> | <s-expr>+ ;
<sig-val>:: "(" <pub-sig-alg-id> <sig-params> ")" ;
<signature>:: "(" "signature" <hash-of-obj> <signer> <sig-val> ")" ;
<relative-name>:: "(" "name" <byte-string> ")" ;
<local-name>:: "(" "name" <principal> <byte-string> ")" ;
<fq-name>:: "(" "name" <principal> <byte-string>+ ")" ;
<name>:: <fq-name> | <local-name> | <relative-name> ;
```

## ACL, Certificate, and Sequence

```
<version>:: "(" "version" <byte-string> ")" ;
<cert-display>:: "(" "display" <byte-string> ")" ;
<issuer-loc>:: "(" "issuer-info" <loc-obj>+ ")" ;
<subject-loc>:: "(" "subject-info" <loc-obj>+ ")" ;
<issuer>:: "(" "issuer" <principal> ")" ;
<k-val>:: {<integer> that's always > 1} ;
<scheme-id>:: <byte-string> ;
<threshold>:: "(" "thresh" <k-val> <scheme-id> ")" ;
```

```
<subj-cap-obj>:: <principal> | <relative-name> | <local-name> ;
<subject-cap>:: "(" "subject" <threshold>? <subj-cap-obj> ")" ;
<subject-role-obj>:: <principal> | <local-name> ;
<subject-role>:: "(" "subject" <threshold>? <subj-role-obj> ")" ;
<subj-name-obj>:: <principal> | <relative-name> | <local-name> ;
<subject-name>:: "(" "subject" <subj-name-obj> ")" ;
<subj-transfer-obj>:: <principal> ;
<subject-transfer>:: "(" "subject" <subj-transfer-obj> ")" ;
<subj-attr-obj>:: <principal> | <relative-name> |
<local-name> | <hash-of-obj> | <uris> ;
<subject-attr>:: "(" "subject" <subj-attr-obj> ")" ;
<subject-acl>:: <principal> | <fq-name> ;
<role>:: "(" "role" <relative-name> ")" ;
<tag-star>:: "(*)" ;
<tag-expr>:: <tag-star> | <s-expr> ;
<tag>:: "(" "tag" <tag-expr>? ")" ;
<capability>:: <tag> ;
<attribute>:: <tag> ;
<redelegate>:: "delegateable" ;
<transfer>:: "transferable" ;
<propagate>:: <redelegate> | <transfer> ;
<grant>:: "(" "grant" <propagate>? ")" ;
<delegate>:: "(" "delegate" <redelegate>? ")" ;
<emp-value>:: <grant> | <delegate> ;
<empower>:: "(" "empower" <emp-value> ")" ;
<escrow-status> :: "verified" | "to-verify" ;
<escrow>:: "(" "escrow" <escrow-status> <loc-obj>? ")" ;
<date>:: {ASCII-encoded <byte-string> "YYYY-MM-DD_HH:MM:SS" in UTC time} ;
<not-before>:: "(" "not-before" <date> ")" ;
<maybe-after>:: "(" "maybe-after" <date> ")" ;
<not-after>:: "(" "not-after" <date> ")" ;
<valid-basic>:: <not-before>? <maybe-after>? <not-after>? ;
<nonce>:: <byte-string> ;
```

```
<valid-one>:: "(" "one-time" <nonce> ")" ;
<new-cert>:: "(" "new-cert" <loc-obj> ")" ;
<off-line>:: "(" "off-line" <valid-one>? <new-cert>? ")" ;
<on-line>:: "(" "on-line" <loc-obj> <principal>? ")" ;
<recency>:: <off-line> | <on-line> ;
<valid>:: "(" "valid" <valid-basic>? <recency>? ")" ;
<valid-transfer>:: "(" "valid" <not-before> ")" ;
<created>:: "(" "created" <date> ")" ;
<comment>:: "(" "comment" <byte-string> ")" ;
<acl-entry>:: "(" "entry" <subject-acl> <capability> <empower>?
<valid>? <comment>? ")" ;
<acl>:: "(" "acl" <version>? <acl-entry>* ")" ;
<role-cert>:: "(" "role-cert" <version>? <cert-display>? <issuer>
<issuer-loc>? <subject-role> <subject-loc>? <empower>
<role> <valid> <escrow>? <created>? <comment>? ")" ;
<capability-cert>:: "(" "cap-cert" <version>? <cert-display>? <issuer>
<issuer-loc>? <subject-cap> <subject-loc>? <empower>
<capability> <valid> <escrow>? <created>? <comment>? ")" ;
<name-cert>:: "(" "name-cert" <version>? <cert-display>? <issuer>
<subject-name> <relative-name> <valid>
<escrow>? <created>? <comment>? ")" ;
<transfer-cert>:: "(" "transfer-cert" <version>? <cert-display>? <issuer>
<issuer-loc>? <subject-transfer> <valid-transfer>
<escrow>? <comment>? ")" ;
<attr-cert>:: "(" "attr-cert" <version>? <cert-display>? <issuer>
<subject-attr> <subject-loc>? <attribute> <valid> <escrow>?
<created>? <comment>? ")" ;
<auth-cert>:: <role-cert> | <capability-cert> ;
<cert-body>:: <auth-cert> | <name-cert> | <transfer-cert> | <attr-cert> ;
<cert>:: "(" "cert" <cert-body> <sig-val> ")" ;
<reduce-op>:: "(" "do" "reduce" <sequence> ")" ;
<general-op>:: "(" "do" <byte-string> <s-part>* ")" ;
<op>:: <reduce-op> | <general-op> ;
```

```
<seq-def-name>:: <hash> | <s-expr> ;
<seq-def-value>:: <pub-key> | <cert> | <sequence> ;
<seq-def>:: "(" "def" <seq-def-name> <seq-def-value> ")" ;
<seq-ref>:: "(" "ref" <seq-def-name> ")" ;
<seq-ent>:: <cert-body> | <cert> | <principal> | <signature> |
<op> | <seq-def> | <seq-ref> | <sequence> ;
<sequence>:: "(" "sequence" <seq-ent>* ")" ;
```

# Appendix C

# Authorization-chain Reduction Examples

To illustrate the *Authorization-chain Reduction Algorithm* presented earlier, I have included three CBAC-based examples as shown in Figure C.1, C.2, C.3 and C.4. To keep it short, however, I decided to keep the capability and validity fields static for all tuples and show only the flow of authorization from principal to principal.

The first example is the reduction of a delegation chain (Figure C.1a) which consist of the following four tuples:

$$\langle K_A, K_B, (delegate, redelegatable), A_1, V_1 \rangle \tag{C.1}$$

$$\langle K_B, K_B\ n, (delegate, redelegatable), A_1, V_1 \rangle \tag{C.2}$$

$$\langle K_B, K_C, K_B\ n, V_1 \rangle \tag{C.3}$$

$$\langle K_C, K_D, (delegate, no-propagation), A_1, V_1 \rangle \tag{C.4}$$

Since there is name-mapping involved, the *Name Reduction Rules* is applied from the top down. This results in the reduction of tuple (C.2) and tuple (C.3) into:

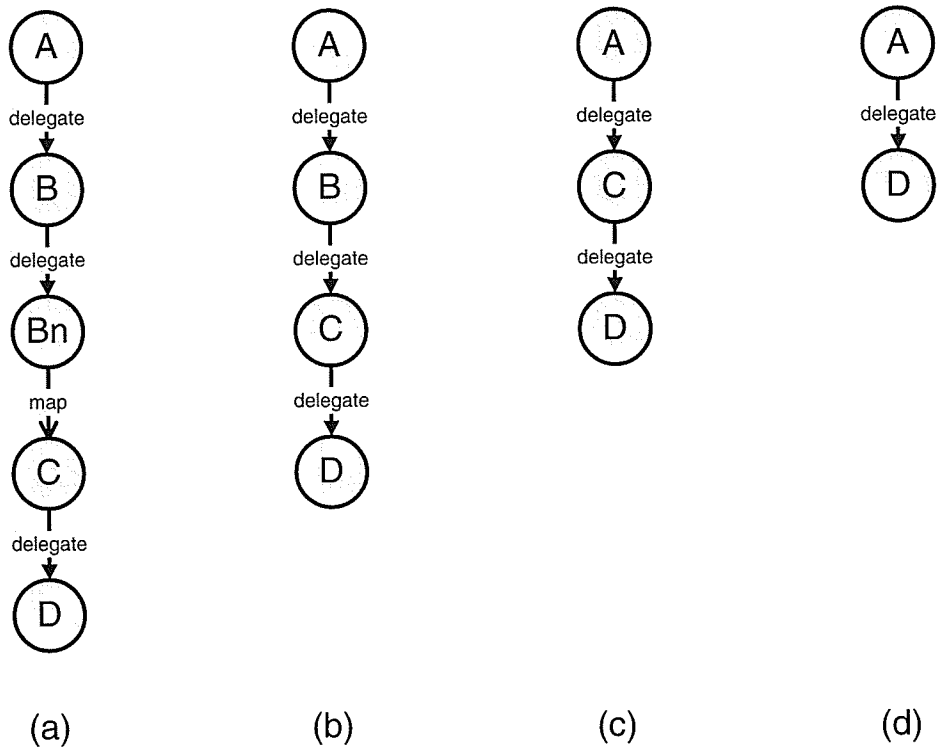


Figure C.1: A hypothetical delegation chain reduction

$$\langle K_B, K_C, (delegate, redelegatable), A_1, V_1 \rangle \quad \text{(C.5)}$$

Now we can apply the *Round-trip Reduction* algorithm, beginning with the *Non-threshold 5-tuples Reduction Rules*, to the chain (C.1),(C.5),(C.4) (Figure C.1b). First, the algorithm encounters tuple (C.1) and tuple (C.5) which are reduced into:

$$\langle K_A, K_C, (delegate, redelegatable), A_1, V_1 \rangle \quad \text{(C.6)}$$

The resulting chain (C.6),(C.4) (Figure C.1c) are then reduced into:

$$\langle K_A, K_D, (delegate, no-propagation), A_1, V_1 \rangle \quad \text{(C.7)}$$

The result is a single 5-tuple (Figure C.1d), so we're finished.

The second example is the reduction of a granting chain (Figure C.2a) which consist of the following four tuples:

$$\langle K_A, K_B\ n, (grant, transferable), A_1, V_1 \rangle \quad \text{(C.8)}$$

$$\langle K_B, K_C, K_B\ n, V_1 \rangle \quad \text{(C.9)}$$

$$\langle K_C, K_D, (transfer), A_1, V_1 \rangle \quad \text{(C.10)}$$

$$\langle K_D, K_E, (transfer), A_1, V_1 \rangle \quad \text{(C.11)}$$

The *Name Reduction Rules* is first applied and thus reducing tuple (C.8) and tuple (C.9) into:

$$\langle K_A, K_C, (grant, transferable), A_1, V_1 \rangle \quad \text{(C.12)}$$

Proceeding with the *Non-threshold 5-tuples Reduction Rules* on the chain (C.12),(C.10),(C.11) (Figure C.2b), the tuple (C.12) and tuple (C.10) are first reduced into:

$$\langle K_A, K_D, (grant, transferable), A_1, V_1 \rangle \quad \text{(C.13)}$$

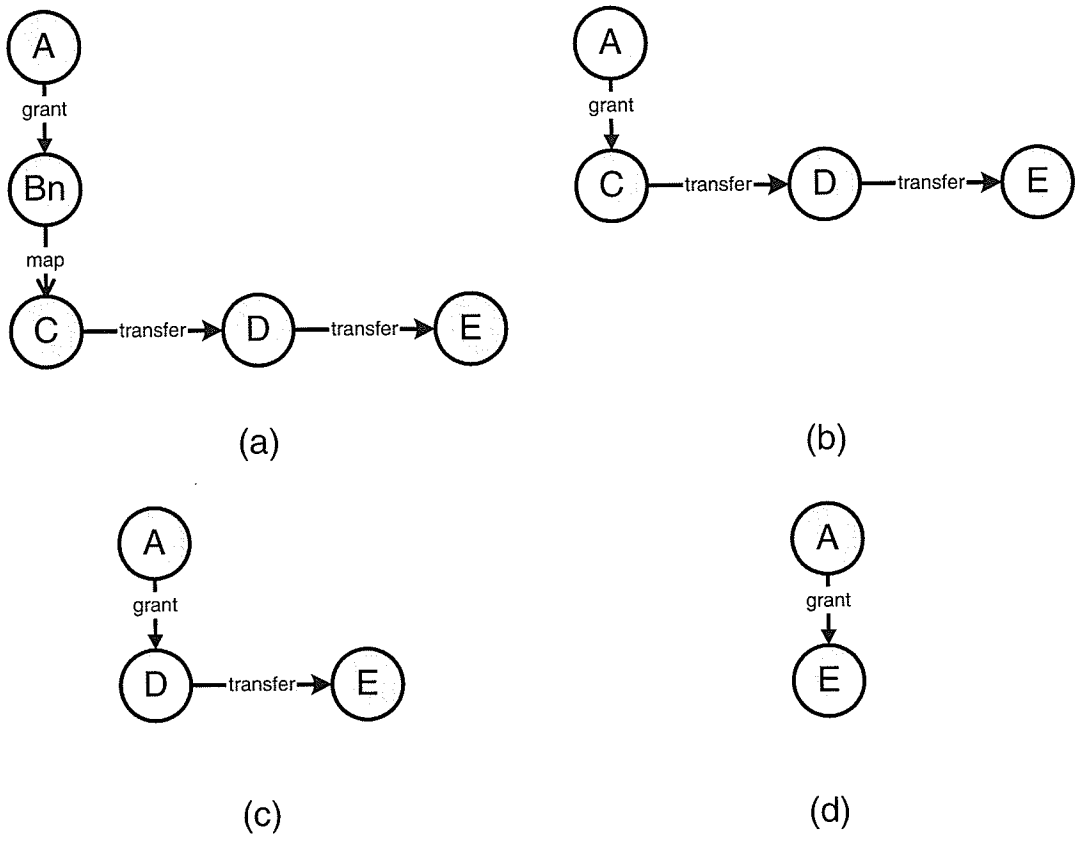


Figure C.2: A hypothetical granting chain reduction

The resulting chain (C.13),(C.11) (Figure C.2c) are then reduced into:

$$\langle K_A, K_E, (grant, transferable), A_1, V_1 \rangle \tag{C.14}$$

Again, the result is a single 5-tuple (Figure C.2d), so we're finished.

The third example is the reduction of a thresholded delegateable granting chain (Figure C.3a) which consist of the following tuples:

$$\langle K_A, K_B, (grant, delegateable), A_1, V_1 \rangle \tag{C.15}$$

$$\langle K_B, (2\ SID_1) K_C, (delegate, delegateable), A_1, V_1 \rangle \tag{C.16}$$

$$\langle K_C, K_D, (delegate, delegateable), A_1, V_1 \rangle \tag{C.17}$$

$$\langle K_D, K_E, (delegate, delegateable), A_1, V_1 \rangle \tag{C.18}$$

$$\langle K_B, (2\ SID_1) K_F, (delegate, delegateable), A_1, V_1 \rangle \tag{C.19}$$

$$\langle K_F, (2\ SID_2) K_G, (delegate, delegateable), A_1, V_1 \rangle \tag{C.20}$$

$$\langle K_G, K_E, (delegate, delegateable), A_1, V_1 \rangle \tag{C.21}$$

$$\langle K_F, (2\ SID_2) K_H\ n, (delegate, delegateable), A_1, V_1 \rangle \tag{C.22}$$

$$\langle K_H, K_E, K_H\ n, V_1 \rangle \tag{C.23}$$

The *Name Reduction Rules* is first applied and thus reducing tuple (C.22) and tuple (C.23) into:

$$\langle K_F, K_E, (delegate, delegateable), A_1, V_1 \rangle \tag{C.24}$$

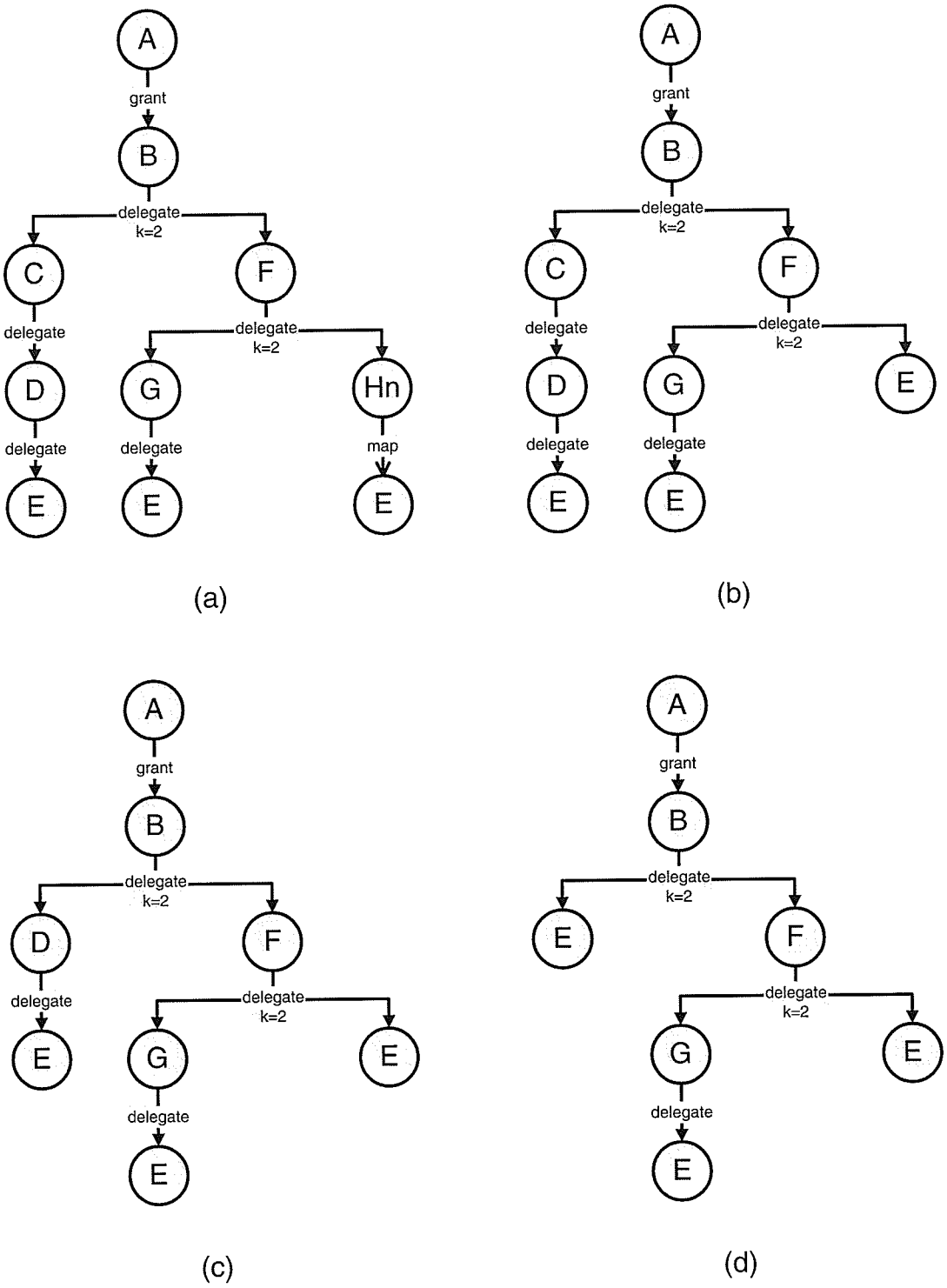


Figure C.3: A hypothetical thresholded granting chain reduction

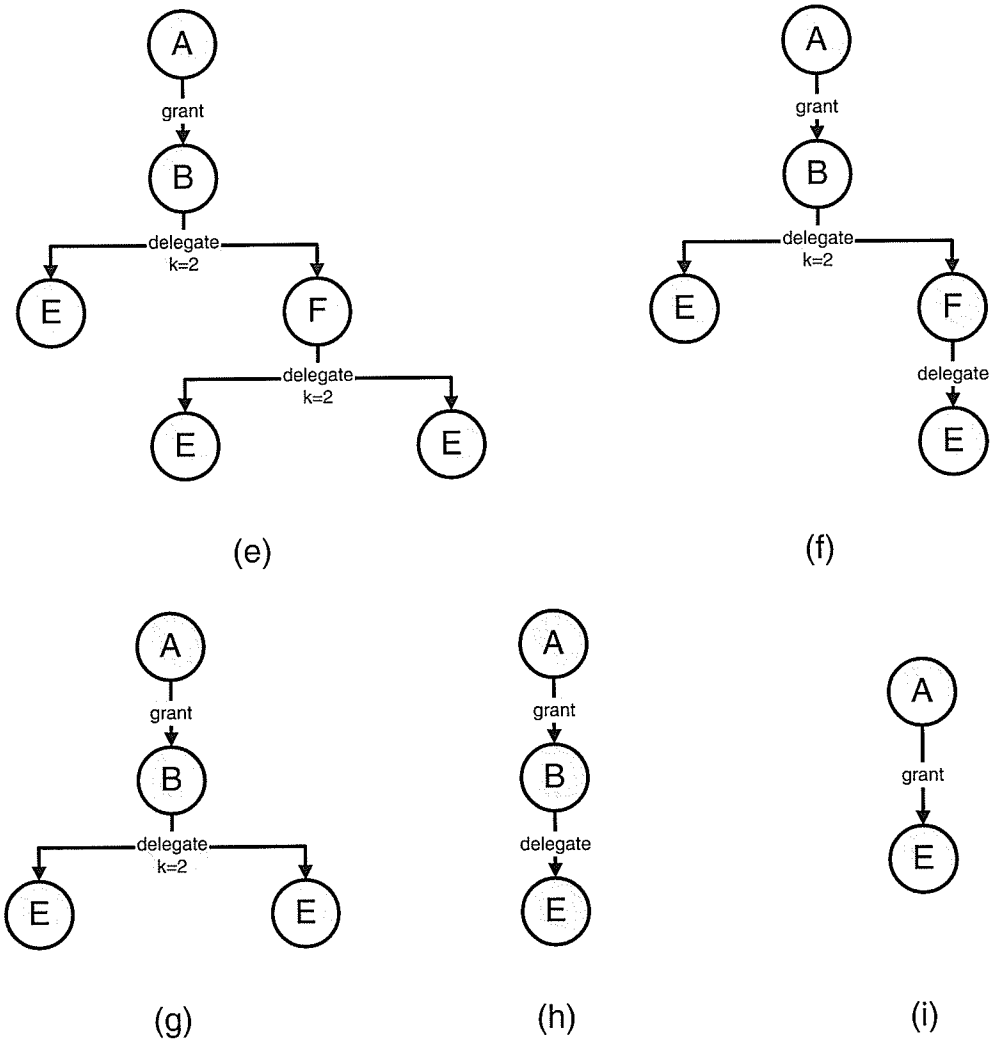


Figure C.4: A hypothetical thresholded granting chain reduction (cont.)

Proceeding with the *Non-threshold 5-tuples Reduction Rules* on the resulting chain (Figure C.3b), tuple (C.16), (C.17), (C.18) and (C.20), (C.21) are reduced (Figure C.3c, d) into the following chain (Figure C.4e):

$$\langle K_A, K_B, (grant, delegateable), A_1, V_1 \rangle \tag{C.25}$$

$$\langle K_B, (2\ SID_1) K_E, (delegate, delegateable), A_1, V_1 \rangle \tag{C.26}$$

$$\langle K_B, (2\ SID_1) K_F, (delegate, delegateable), A_1, V_1 \rangle \tag{C.27}$$

$$\langle K_F, (2\ SID_2) K_E, (delegate, delegateable), A_1, V_1 \rangle \tag{C.28}$$

$$\langle K_F, (2\ SID_2) K_E, (delegate, delegateable), A_1, V_1 \rangle \tag{C.29}$$

Since the result still contains more than one tuple, we need to apply the *Threshold 5-tuples Reduction Rules* which continues to reduce the chain from the bottom up (Figure C.4f, g, h) into:

$$\langle K_A, K_E, (grant, delegateable), A_1, V_1 \rangle \tag{C.30}$$

Finally, the result is a single 5-tuple (Figure C.4i), so we're finished.

# Appendix D

# Implementation Source Code and Documentation

The attached CD-ROM contains the following:

- Java source code for the *spki*, *spki.interfaces* class libraries;
- Java source code for the *spki.test*, *spki.benchmark* and *spki.demo.run* programs;
- Java source code for the modified version of Myrvang's *sexp* and *database* class libraries;
- Compiled bytecodes for all of the above classes in *.jar* format;
- API documentation for all of the above classes in JavaDoc-style HTML files;
- The *Cryptix* 3.2, the GNU *regexp* 1.0.6 and the GNU *getopt* 1.0.5 class libraries used by the implementation;
- Raw outputs of the *java.benchmark* program executions used for performance benchmarking of the library.

# Acknowledgements

In no particular order:

- Prof. Maheswaran, my advisor.
- TR*Labs*, my sponsor.
- Jose and Alice Rueda, my great friends.
- Canadian, Thai and international friends in Winnipeg.
- Dr. Thaweesak Koanantakool and Apinetr Unakul, my mentors.
- Mom, Dad, Lovely, Dolly and Nilu, my family.
- Rashy, my love.

# Bibliography

[1] Bruce Schneier, *Secrets & Lies: Digital Security in a Networked World*, 1st Edition, John Wiley and Sons, USA, 2000.

[2] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd Edition, John Wiley and Sons, USA, 1996.

[3] T. Stabell-Kulø, F. Dillema, T. Fallmyr, "The Open-End Argument for Private Computing," *Proc. International Symposium on Handheld and Ubiquitous Computing (HUC 99)*, (Karlsruhe, Germany), September 1999.

[4] Roger Clarke, "Conventional Public Key Infrastructure: An Artefact Ill-Fitted to the Needs of the Information Society," November 2000. Prepared for submission to the *IS in the Information Society* track of the *European Conference in Information Systems (ECIS 2001)*, http://www.anu.edu.au/people/Roger.Clarke/II/PKIMisFit.html.

[5] W. Diffie, M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, November 1976, pp. 644–654.

[6] Loren M. Kohnfelder, *Towards a practical Public-key Cryptosystem*, Bachelor's thesis, Massachusetts Institute of Technology, 1978.

[7] ITU-T/ISO/IEC, "ITU-T Reccomendation X.509 — ISO/IEC 9594-8: "Information Technology - Open Systems Interconnection - The Directory: Public-Key and Attribute Certificate Frameworks," 2000.

[8] Y. Hsu, S. Seymour, "An Intranet Security Framework Based on Short-lived Certificates," *IEEE Internet Computing*, Vol. 2, No. 2, March/April 1998.

[9] E. Belani, A. Vahdat, T. Anderson, and M. Dahlin, "The CRISIS Wide Area Security Architecture," *Proc. USENIX Security Symposium, San Antonio, Texas*, January 1998. http://now.cs.berkeley.edu/WebOS/papers/uss.ps.

[10] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, "A Security Architecture for Computational Grids," *Proc. 5th ACM Conference on Computer and Communications Security*, 1998, pp. 83–92. ftp://ftp.globus.org/pub/globus/papers/security.pdf.

[11] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch, "Design and Deployment of a National-Scale Authentication Infrastructure," *IEEE Computer*, Vol. 33, No. 12, December 2000.

[12] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen, "SPKI Certificate Theory," RFC 2693, September 1999. http://www.ietf.org/rfc/rfc2693.txt.

[13] "Public-Key Infrastructure (X.509) (pkix) Charter," http://www.ietf.org/html.charters/pkix-chater.html.

[14] A. Arsenault, S. Turner, "Internet X.509 Public Key Infrastructure," Internet-Draft (work in progress): draft-ietf-pkix-roadmap-06, November 2000.

[15] R. Housley, W. Ford, W. Polk, D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile," Internet-Draft (work in progress): draft-ietf-pkix-new-part1-03, November 2000.

[16] S. Santesson, W. Polk, P. Barzin, M. Nystrom, "Internet X.509 Public Key Infrastructure Qualified Certificates," Internet-Draft (work in progress): draft-ietf-pkix-qc-06, August 2000.

[17] D. Pinkas, T. Gindin, "Internet X.509 Public Key Infrastructure Permanent Identifier," Internet-Draft (work in progress): draft-ietf-pkix-pi-01, August 2000.

[18] S. Farrell, R. Housley, "An Internet Attribute Certificate Profile for Authorization," Internet-Draft (work in progress): draft-ietf-pkix-ac509prof-06, August 2000.

[19] "Simple Public Key Infrastructure (spki) Charter," http://www.ietf.org/html.charters/spki-charter.html.

[20] C. Ellison, B. Schneier, "Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure," *Computer Security Journal*, Vol. 16, No. 1, 2000. http://www.counterpane.com/pki-risks.html.

[21] Carl Ellison, "Naming and Certificates," *Proc. 10th Conference on Computers, Freedom and Privacy (CFP2000)*, April 2000. http://www.cfp2000.org/papers/ellison.pdf.

[22] Carl Ellison, "SPKI Requirements," RFC 2692, September 1999. http://www.ietf.org/rfc/rfc2692.txt.

[23] Ronald Rivest, "Can We Eliminate Certificate Revocation Lists?," *Proc. Financial Cryptography*, 1998. http://theory.lcs.mit.edu/~rivest/publications.html.

[24] T. Aura, C. Ellison, "Privacy and accountability in certificate systems," Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, April 2000. http://www.tcs.hut.fi/Publications/reports/A61abstract.html.

[25] J. Howell, D. Kotz, "A formal semantics for SPKI," *Proc. 6th European Symposium on Research in Computer Security (ESORICS 2000)*, 2000, pp. 140–158. http://www.cs.dartmouth.edu/~jonh/research/delegation/esorics-abs.pdf.

[26] Ninghui Li, "Local Names In SPKI/SDSI," *Proc. 13th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, July 2000, pp. 2–15. http://cs1.cs.nyu.edu/ninghui/papers/csfw13.ps.

[27] Tuomas Aura, "Distributed Access-Rights Management with Delegation Certificates," in *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, J. Vitek, C. Jensen, ed., Vol. 1603 of *LNCS*, Springer-Verlag, Germany, 1999, pp. 211–235. http://www.tcs.hut.fi/Publications/papers/aura/aura-lncs1603-abstract.html.

[28] Lynn Wheeler, "Account Authority Digital Signature Introduction," http://www.garlic.com/~lynn/aadsover.htm.

[29] A. Wheeler, L. Wheeler, "PKI Account Authority Digital Signature Infrastructure," Internet-Draft (work in progress): draft-wheeler-ipki-aads-01, November 1998.

[30] "ANSI Online Store," http://store.ansi.org/.

[31] R. Sandhu, "Role-Based Access Control Models," in *Advances in Computers*, Vol. 46, Academics Press, USA, 1998.
http://www.list.gmu.edu/articles/advcom/a98rbacps.

[32] E. Barka, R. Sandhu, "Framework for Role-Based Delegation Models," *16th Annual Computer Security Applications Conference*, December 2000.
http://www.acsac.org/2000/abstracts/34.html.

[33] Roger Clarke, "Privacy Requirements of Public Key Infrastructure," *IIR IT Security Conference*, March 2000.
http://www.anu.edu.au/people/Roger.Clarke/DV/PKI2000.html.

[34] Roger Clarke, "Chip-Based ID: Promise and Peril," *International Conference on Privacy*, (Montreal), September 1997.
http://www.anu.edu.au/people/Roger.Clarke/DV/IDCards97.html.

[35] Tuomas Aura, "On the Structure of Delegation Networks," *Proc. 11th IEEE Computer Security Foundations Workshop*, 1998.

[36] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen, "Simple Public Key Certificates," Internet-Draft (work in progress):
draft-ietf-spki-cert-structure-06, July 1999.

[37] T. Aura, D. Gollmann, "Software license management with smart cards," *Proc. USENIX Workshop on Smartcard Technology*, (Chicago), USENIX Association, May 1999, pp. 75–85.

[38] Ronald L. Rivest, "SEXP (S-expressions),"
http://theory.lcs.mit.edu/~rivest/sexp.html.

[39] Carl Ellison, "ASN.1 Misuse," *RSA Conference*, 1996.
http://world.std.com/~cme/html/asn1.ps.

[40] D. Clarke, J. Elien, C. Ellison, M. Fredette, A. Morcos, R. Rivest, "Certficate Chain Discovery in SPKI/SDSI," September 2001. Draft paper; To appear in *Journal of Computer Security*, http://theory.lcs.mit.edu/~rivest/publications.html.

[41] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen, "SPKI Examples," Internet-Draft (work in progress): draft-ietf-spki-cert-examples-01, March 1998.

[42] "The Java-Powered iButton," http://www.ibutton.com/.

[43] National Institute of Standards and Technology (NIST), "Federal Information Processing Standards Publication: Security Requirements for Cryptographic Modules FIPS 140-2 (Draft)," November 1999. http://csrc.nist.gov/publications/fips/.

[44] National Institute of Standards and Technology (NIST), "Federal Information Processing Standards Publication: Security Requirements for Cryptographic Modules FIPS 140-1," January 1994. http://csrc.nist.gov/publications/fips/.

[45] "ISO - International Organization for Standardization," http://www.iso.org/.

[46] M. Freudenthal, S. Heiberg, J. Willemson, "Personal Security Environment on Palm PDA," *16th Annual Computer Security Applications Conference*, December 2000. http://www.acsac.org/2000/abstracts/33.html.

[47] Peter Gutmann, "An Open-source Cryptographic Coprocessor," *9th USENIX Security Symposium*, 2000. http://www.cryptoengines.com/~peter/usenix00.pdf.

[48] J. Kelsey, B. Schneier, "Authenticating Secure Tokens Using Slow Memory Access," *Proc. USENIX Workshop on Smart Card Technology*, 1999, pp. 101–106. http://www.counterpane.com/slow-memory.html.

[49] Per Harald Myrvang, *An Infrastructure for Authentication, Authorization and Delegation*, Master's thesis, University of Tromsø, May 2000. http://www.pasta.cs.uit.no/thesis/perm.html.

[50] "Cryptix 3," http://www.cryptix.org/products/cryptix31/.