Security System for Ad-hoc Wireless Networks based on Generic Secure Objects

Matei Ciobanu Morogan

2005

Ph.D. Thesis
Kungliga Tekniska Högskolan

Submitted as the partial fulfillment of requirements for the degree of Doctor of Philosophy at the Department of Computer and System Sciences, Kungliga Tekniska Högskolan
Doctoral Dissertation
Department of Computer and System Sciences
Stockholm University / Royal Institute of Technology
ISBN 91-7178-019-X
Abstract

As computing devices and wireless connectivity become ubiquitous, new usage scenarios emerge, where wireless communication links between mobile devices are established in an ad-hoc manner. The resulting wireless ad-hoc networks differ from classical computer networks in a number of ways, lack of permanent access to the global network and heterogeneous structure being some of them. Therefore, security services and mechanisms that have been designed for classical computer networks are not always the optimal solution in an ad-hoc network environment.

The research is focused on analyzing how standard security services that are available in classical networks can be provided in an ad-hoc wireless network environment. The goal is to design a security system optimized for operation in ad-hoc wireless networks that provides the same security services – authentication, access control, data confidentiality and integrity, non-repudiation – currently available in classic wired networks.

The first part of the thesis is the design and implementation of a security platform based on generic secure objects. The flexible and modular nature of this platform makes it suitable for deployment on devices that form ad-hoc networks – ranging from Java-enabled phones to PDAs and laptops.

We then investigate the problems that appear when implementing in ad-hoc networks some of the security technologies that are standard building blocks of secure systems in classical computer networks. Two such technologies have been found to present problems, namely the areas of certification and access control. In a series of articles, we have described the problems that appear and devised solutions to them by designing protocols, techniques and extensions to standards that are optimized for usage in the ad-hoc network environment.

These techniques, together with the functionality provided by the underlying security platform, are used to implement all standard security services – confidentiality, authentication, access control, non repudiation and integrity, allowing to integrate ad-hoc networks into the existing security infrastructure.
Acknowledgements

I would like to take this opportunity to express my gratitude to all the people at DSV for a great environment during the entire time I have worked on this thesis.

Most notably, heartfelt thanks to my two supervisors: to Sead Muftic for his guidance, ideas and comments, and to Louise Yngström for her great support and encouragement. To the group at K2LAB that provided many interesting ideas on mobile applications: Carl-Gustaf Jansson, Fredrik Kilander, Patrik Werle, Martin Jonsson and all the others. My thanks to my fellow doctoral candidates Jeffy Mwakalinga and Kasun DeZoysa; and to my colleagues and friends Anette Hulth, Fresia Pérez, Atelach Alemu and Christian Schulte for their fascinating insights.
Contents

CHAPTER 1: INTRODUCTION ................................................................. 1
  1.1 BACKGROUND ............................................................................. 2
  1.2 SECURITY ISSUES ....................................................................... 3
  1.3 RESEARCH FOCUS AND LIMITATIONS ...................................... 4
  1.4 RELATED RESEARCH ................................................................. 6
  1.5 RESEARCH METHOD ................................................................. 9
  1.6 RESEARCH RESULTS ................................................................. 9
    1.6.1 Part 1: Modular Security Platform based on Generic Secure Objects .......... 9
    1.6.2 Part 2: Certificate Management in Ad Hoc Networks ...................... 10
    1.6.3 Part 3: Access Control for Documents in Ad-hoc Networks ................. 11
  1.7 VALIDATION OF RESULTS ...................................................... 13

PART 1: GENERIC SECURITY OBJECTS - A COMPREHENSIVE JAVA
SECURITY SYSTEM FOR OPEN NETWORKS .................................. 15

CHAPTER 2: SECURITY OBJECTS – CONCEPTS AND PRINCIPLES......... 19
  2.1 INTRODUCTION .......................................................................... 19
  2.2 ADVANTAGES OF SECURITY OBJECTS ................................... 20
    2.2.1 Security Functionality .......................................................... 20
    2.2.2 Compatibility and Scaling ................................................... 20
    2.2.3 Rapid development and broader applicability ............................ 21
    2.2.4 Commercial Off-The Server Systems ................................... 22
    2.2.5 Evaluation and Validation .................................................... 22
  2.3 RELATED RESEARCH ............................................................... 23
  2.4 APPLICATION SCENARIOS ........................................................ 24

CHAPTER 3: SPECIFICATION OF GENERIC SECURITY OBJECTS ........ 31
  3.1 IDENTIFICATION AND AUTHENTICATION OBJECTS .................. 34
  3.2 CRYPTOGRAPHIC KEY OBJECTS ............................................. 38
  3.3 CERTIFICATION OBJECTS ....................................................... 38
  3.4 SECURE SESSION OBJECTS .................................................... 41
  3.5 ACCESS CONTROL AND AUTHORIZATION OBJECTS ................. 42
  3.6 SMART CARD OBJECTS .......................................................... 43

CHAPTER 4: PROTOCOLS AND USAGE OF SECURITY OBJECTS .......... 45
  4.1 LOCAL LOGIN ............................................................................. 45
8.2.2 Access control process ................................................................. 98
8.2.3 Security of the system ................................................................. 99
8.3 Access control for PKCS#7 Documents ............................................ 99
  8.3.1 Extending the PKCS#7 standard .................................................. 100
  8.3.2 Access control process ............................................................... 102
  8.3.3 Security of the system ............................................................... 103
8.4 Conclusions ................................................................................. 104

CHAPTER 9: ENFORCING ACCESS CONTROL POLICIES IN AD-HOC NETWORKS ......................................................................................................................... 107
9.1 Introduction .................................................................................. 107
  9.1.1 Model of the access control system ............................................. 107
9.2 The Trusted Decision Engine .......................................................... 109
9.3 Conclusions ................................................................................ 111

CHAPTER 10: CONCLUSIONS AND FUTURE DIRECTIONS OF RESEARCH ................................................................................................................................. 113
10.1 Conclusions ................................................................................ 113
10.2 Further Research ....................................................................... 114
  10.2.1 Moving to a Component Based Design .................................... 114
  10.2.2 Standardising the protocols ..................................................... 115
  10.3.3 Deployment of the complete system ....................................... 115

APPENDIX: GENERIC SECURITY OBJECTS ................................................................................................................................. 117
A.1 Identification and Authentication Objects ...................................... 117
  A.1.1 DistinguishedName ................................................................. 117
  A.1.2 UserIdentity ........................................................................ 118
  A.1.3 AuthenticationObject ............................................................ 119
  A.1.4 Login object .......................................................................... 122
  A.1.5 UserRegistration Object ....................................................... 123
A.2 Cryptographic Key Objects .......................................................... 124
  A.2.1 AsymmetricKeyPair ............................................................... 124
  A.2.2 PublicKey ........................................................................... 125
  A.2.3 PrivateKey ........................................................................... 125
  A.2.4 SymmetricKey ...................................................................... 126
A.3 Certification Objects ................................................................... 128
  A.3.1 Certificate ............................................................................ 128
  A.3.2 CertificateChain ................................................................. 130
  A.3.3 CertificateRequest ............................................................... 131
  A.3.4 CertificationClient ............................................................... 131
  A.3.5 CRL .................................................................................... 132
A.4 Secure Session Objects .............................................................. 133
  A.4.1 EncapsulatedObject ............................................................. 133
  A.4.2 SecureSession ................................................................. 136
A.5 Access Control and Authorization Objects .................................. 137
  A.5.1 Capability ........................................................................... 137
  A.5.2 Attribute Certificate ......................................................... 137
Chapter 1: Introduction

As computing devices and wireless connectivity become ubiquitous, new usage scenarios emerge, where wireless communication links between mobile devices are established in an ad-hoc manner. The resulting wireless ad-hoc networks differ from classical computer networks in a number of ways, lack of permanent access to the global network and heterogeneous structure being some of them. Therefore, security services and mechanisms that have been designed for classical computer networks are not always the optimal solution in an ad-hoc network environment.

The standard security services for computer networks have been well defined and researched. The ISO standard defines five such services: Privacy, Authentication, Access Control, Non-Repudiation and Availability. The existing implementations of these services are generally optimized for a classical network environment. These solutions do not always work, or may be very inefficient in ad-hoc wireless environments. The result is that communications in ad-hoc networks is either unsecure or poses stringent limitations on the functionality of the networks, for example by requiring access to an on-line server for specific transactions.

Our research is focused on analyzing how standard security services that are available in classical networks can be provided in an ad-hoc wireless network environment. The goal is to design a security system optimized for operation in ad-hoc wireless networks that provides the same security services – authentication, access control, data confidentiality and integrity, non-repudiation – now available in classic wired networks.

Not being dependent on an existing Internet infrastructure, ad-hoc network techniques can be more robust than the corresponding techniques used in wired networks. This robustness can make them useful even in the classical network scenario, even though in certain situations it can come with a price in overhead and load.
In the remains of this chapter we will give a background on ad-hoc wireless networks (section 1.1), followed by the security issues related to them in section 1.2. The focus of our research is defined in section 1.3, followed by an overview of the related research area in section 1.4. The methods used for the research and for finding the solutions are described in section 1.5. Finally, section 1.6 presents a short overview of each part of the thesis.

1.1 Background

An ad-hoc wireless network is a network created dynamically by two or more users using mobile devices. Configuration, topology and functionality of a network are based on the proximity of users, since users present in a certain area constitute a particular network [Johnson 1994, Meier 2001]. In such environments especially interesting are group applications, where users dynamically join, establish, and leave an application group [Prakash 1998]. Networks and groups are established and terminated dynamically.

Ad-hoc wireless networks differ from wired local area networks in several important aspects [Roman 1997]. One aspect is that access to the global network can not always be established, either because of lack of coverage or because of high cost of communications. Therefore, transactions and control mechanisms in such networks must be as independent from outside resources as possible. Another specific aspect is unpredictability of the network, that is, devices which form it are not pre-registered or known in advance. A network is formed dynamically by users and devices in proximity. For instance, when several people meet in a public place, a network would be formed by their PDA:s, mobile phones, laptops, and maybe some public terminal in their vicinity. Finally, the control of devices in the network can not be as strict as in wired LANs. In wireless networks it is quite easy to connect from the outside without knowledge of other users, since proximity is quite difficult to specify and use as an access control criteria.

In conclusion, the following aspects are key differences between ad-hoc proximity networks and classic, wired LANs:

- Access to servers or resources in the global network may be expensive or impossible.
- Network configuration is unpredictable, due to unknown and unregistered users.
- Access to a network is open to all devices and users in physical proximity.
- Devices that form an ad-hoc network, ranging from mobile phones to laptop computers and networked speaker systems, are usually heterogeneous with limited computing and memory resources.
- Devices that form ad-hoc networks are more exposed to failure and theft than wired stationary computers.

1.2 Security Issues

Because of specific characteristics of formation and usage of ad-hoc proximity networks, security for such networks is more important and also not completely equivalent to security in standard wired LANs.

We can broadly define three groups of security issues for ad-hoc proximity networks. The first is that standard, low-level network protocols, like routing, node and service discovery, multicast and broadcast messages, would not work, or would be very vulnerable in mobile ad-hoc networks due to the dynamic and collaborative nature of this environment.

The second is that classical security services and mechanisms for computer networks may not work in ad-hoc networks or may pose constraints that highly reduce their functionality. Due to openness of wireless networks, there is also a need to use security at much larger scale, since devices are no longer in a safe environment protected by other security solutions (e.g. firewalls, steel locks).

The third group of issues emerges from the fact that ad-hoc proximity networks provide different kinds of usage and access to computing devices which do not exist in classical networks (for example, users sharing a screen/panel or other resources). These usages require novel security solutions and protocols different from the ones already existing.

Security issues related to the first group are:

- Routing protocols, dealing with malicious and / or selfish node behaviour in multi-hop routing.
- Broadcast / multicast message security
- Discovery of services: in a wired network, a computer can broadcast an inquiry for a certain service (i.e. a name server). Since the configuration of the network is controlled by the administration, the response it gets is usually not subjected to a very strong control. In a wireless ad-hoc network however it can no longer be asserted that all parties in the network are “friendly”.

The following security issues are important in the second group:

- Secure server-less protocols and other forms of peer-to-peer operations: the parties in an ad-hoc network may not be able to access certain security
servers/security services on the Internet for some periods of time, but security should still function within the network.
- Protection of data stored in devices: devices are much more likely to be stolen or physically damaged than stationary computers in a network.

Some issues in the third group are:

- Privacy control (a computer may leave a trace which interferes with the user’s privacy)
- More flexible and fine-grained authorization system, as the complexity of the access decisions increases, depending on user domain, physical location, peer user’s domain and so on.
- Various forms of secure (group) transactions between parties.

The list of issues in all three groups is not exhaustive. They represent just some of the issues that need to be addressed in a security system for ad-hoc wireless networks. Other issues may also become significant, especially in the third group, as the networks evolve with new devices and applications. As new functions and applications are introduced, specific security mechanisms and solutions will be needed to secure them.

1.3 Research focus and limitations

A security system for ad-hoc networks has to be based on different requirements than systems for classical networks and take into account specific limitations of ad-hoc networks, as outlined in the previous section.

The area that is investigated in this thesis is implementation of standard security services such as authentication, authorization, access control and non-repudiation in an ad-hoc network environment, that is, to provide solutions to the second group of security issues as they are described in section 1.2.

Furthermore, we need to specify the type of scenario these solutions are designed for. The term mobile ad-hoc networks currently encompasses a large variety of situations, the main differences between them being related to device power and connectivity, and the security solutions need to be adapted for the respective scenario. One such example is sensor networks, where devices have very little computing and battery power, and have no other connection than one node to another. The scenario this thesis focuses on is a network made of devices with varying computational capabilities – e.g. PDAs, laptops, mobile phones, various media devices – that can connect to each other in a dynamic manner. On-line access may be very expensive for the entities in the ad-hoc network (e.g. cellular data network) or it may not be available at all [Roman 1997], but they – or at
least some of them – also get connected to the network infrastructure occasionally.

The goal of this research is to find conceptual solutions to the specific security issues of mobile ad-hoc networks and then to design a security system based on these principles. The scope of the design is to create security mechanisms, services and protocols that are optimized for usage in an ad-hoc wireless network environment as the one described above. The design will be based on the concept of generic secure objects [Muftic 2001], which has been shown to be particularly suited to the requirements of such environments [Morogan 2000].

Our approach can be seen as a layered model (Figure 1.1). The lowest layer is the security platform – a collection of objects implementing basic security functionality such as encryption, digital signatures and encapsulation. The middle layer deals with the specific protocols and functionalities of the system, and is based on the bottom layer. On the top, there is the policy of the system, a set of rules that decide the operation of the protocols in different situations.

For the bottom layer of our pyramid, we have designed a security platform based on generic secure objects (GSO). These are software objects with specific security functionalities that can be used in various contexts to implement security services. They are particularly suitable for designing and implementing security services for ad-hoc networks, due to their small size, mobility, portability, high flexibility, and functionality. Security functions can easily be implemented by using objects in different configurations. Also, generic secure objects simplify the deployment of the system on different hardware platforms with different operating systems and computing resources.

The second layer of the model provides standard network security services and protocols for the wireless ad-hoc environment. By analyzing the characteristics of these networks outlined in section 1.1, we have identified two security areas where the techniques used in classical networks present problems in wireless ad-hoc networks. These areas are certificate management and access control, where the lack of on-line access makes the existing solutions unworkable or inefficient. In a series of articles, we have investigated problems related to these two areas, and we have proposed solutions to optimize them for this particular environment. The resulting techniques, together with the functionality provided by the underlying GSO platform, are used to implement all standard ISO security services – confidentiality, authentication, access control, non repudiation and integrity.
The third layer contains the security policies that define the operation of the system. The solutions we propose require a security policy that is more flexible than what is necessary in classical networks. To improve the efficiency of the security services, a balance between risk and usability has to be specified in the security policy for each particular situation. However, there are many existing systems and results (a good overview in [SCC 1997] and [Sterne 1991]) that can accommodate complex and flexible security policies. System policy is therefore not within the scope of our work. It could be an interesting direction for further research to study the implementation and design of security policies optimized for ad-hoc networks.

1.4 Related research

The area of security in wireless ad-hoc networks has been in expansion during the last years. It is now possible to identify several sub-areas that have received lots of attention lately.


Yet another area of research is related to security issues specific to the environment of wireless ad-hoc networks, mainly various group communication schemes ([Kaya 2003, Mäki 2000, Yasinsac 2002, Prigent 2003]). Related to group communication is the issue of key management and distribution in ad hoc networks. Several papers ([Asokan 2000, Carman 2002, Hietalahti 2001, Khalili 2003, Zhu 2003b]) propose various key distribution schemes suitable for use in this environment.

The problems related to intrusion detection in ad hoc networks are described in [Brutch 2003], while some solutions are proposed in [Huang 2003] and [Kachirski 2002].

The area that is the current focus of our research regards the implementation in the environment of wireless ad-hoc networks of security services that exist in classical wired networks. Such services include authentication, certification, authorization and access control. The differences between the two environments ([Roman 1997]) make implementation of these services a less than trivial matter.

Various solutions have been suggested for authentication and certification in ad-hoc wireless networks. One of the most difficult problems with using certificates for authentication in ad-hoc networks is related to the revocation of existing certificates. [Eschenauer 2002a] discusses the problem of certification in ad hoc networks versus the Internet, and offers some solutions to establish trust without using certificates. [Balfanz 2002] discusses different protocols for authentication in ad-hoc networks, also without the use of certification, but based on transmitting authentication information through a separate link (IR or contact between the nodes). [Hubaux 2001] describes a system for ad hoc networks similar to the PGP certification system ([Zimmerman 1995]) where certificates are issued, stored and distributed by the users. [Burmeister 1999] presents a theoretical model for using certificates and multiple paths of verification. An approach where the IP address of a node is calculated from its public key is presented in [Montenegro 2002]. [Venkatraman 2000] proposes an authentication scheme based on certificates, without mentioning the issues related to certificate revocation. [Candolin 2002] is also based on certificates, but here the nodes in the network may declare that other nodes have been compromised, thereby providing some form of certificate revocation, raising instead the issue of node misbehaviour and possibility of denial-of-service attacks based on maliciously accusing other nodes. Another solution based on delegating the revocation decision to the nodes is presented in [Crepeau 2003]. The authors here try to make malicious accusations more difficult, but the scheme still rests on trusting a network of nodes, and the revocation decision is still taken by the nodes, instead of a central authority.
Other solutions for certification in ad-hoc networks are based on the idea of threshold cryptography ([Shamir 1979, Chor 1985, Herzberg 1995, Shoup 2000]). [Zhou 1999] proposes using threshold cryptography for distributing the responsibility of a CA to several nodes in the ad-hoc network. Various solutions based on this idea are proposed and implemented in [Kong 2001], [Kong 2002], [Luo 2002] and [Yi 2003]. The weakness of schemes based on threshold cryptography is that on one hand they are computationally intensive, and on the other they rely on the cooperation of nodes in the network. Another problem is that for many applications, the enforcement of central policy decisions is necessary, thereby making a cooperative CA unsuitable.

Our approach in the area has been to rely on an existing CA infrastructure in the wired network, using certificates obtained on-line, and to find techniques for distributing revocation lists through ad hoc networks. The techniques that we suggest are peer-to-peer distribution of CRLs between nodes and distribution points for all CRLs in a domain ([Morogan 2003b] and [Morogan 2003c]).

In the area of authorization and access control, there are some interesting research results in distributed authorization systems which could be used in wireless ad hoc networks. [Johnston 1996] proposes a system based on use-conditions certificates for the protected resources and credential certificates for users. A policy engine compares the two and issues a ticket for the resource access controller. Another paper ([van Doorn 1996]) describes the design and implementation of secure network objects, and provides security for object-oriented network communication. The design accommodates both ACLs [Lampson 1974] and capabilities [Dennis 1966].

[Thompson 1999] describes the implementation of an access control system based on use-condition certificates for the resources and attribute certificates for the users. In this system, resources reside or are represented by servers. To access them, a client connects to the specific server and authenticates herself. The server then fetches the access policies, in the form of Use-Condition certificates, from their prespecified locations, then checks user’s attribute certificates to see if they fit the specified conditions.

The XML Access Control Language (XACL) provides a way to embed access control information in XML documents ([Hada 2000]), and [Damiani 2000] present an access control system that allows definition of access restrictions inside the XML documents.

All the above systems use a form of capabilities on the user side and conditions on the resource side, and as a result they are very flexible. They are also based on accessing a server trusted by the security administration, and that makes access
control decisions, an approach that due to its reliance on centralized network resources is not suitable for ad-hoc networks.

Our research in the area of access control in ad hoc networks has concentrated on the issue of documents residing on the devices that form the network. In a wired, centralised network these documents would reside on a secure server which would enforce the access control policy for each document. In an ad hoc network however access to a central server is not possible, and peer-to-peer transmission of documents has to be used instead. To enforce an access control policy even in this case, we choose to have the documents cryptographically encapsulated at all times and only readable by special trusted decision engines, hardware devices (e.g. smart cards) that can take access control decisions on behalf of the domain authority ([Morogan 2003d, Morogan 2004]).

1.5 Research Method

The method we employed in our research was to study existing implementations of the security services, and analyze the problems that would arise by applying them in ad-hoc wireless networks. Once such problems were identified, workarounds were sought. After finding a workable solution to a problem, it was analyzed from the point of view of security, efficiency in the ad-hoc environment, and compatibility with the classical network environment and existing standards. The research is goal-oriented in the meaning that the sought results are software implementations or protocol standards that provide the necessary security functionality in the ad-hoc network environment. These results are further tested by releasing them for review to the security community, and by creating test implementations.

1.6 Research Results

In this section we will present a short summary and the contributions of each part of the thesis.

1.6.1 Part 1: Modular Security Platform based on Generic Secure Objects

The first part of our work presents the design and implementation of a security system based on Generic Secure Objects (GSO). The concept is to have a number of objects that provide advanced security functionality in a modular form. They are designed to be transparently interoperable, so that new functionality can be obtained by combining existing GSOs. The system is a platform on which higher level security services can be implemented.

The advantages of this approach are modularity, ease of use and flexibility. The system is modular, since only the required objects for a specific application are
necessary – something that is important in systems with limited resources. The objects provide a high level of abstraction, making it easy to implement new security functionality and to use this functionality in general applications, therefore making application development quicker and less error-prone. Finally, new functionality can be created as needed by combining the basic objects, making for a flexible system.

Apart from basic security functionality, GSOs provide high-level security functions, such as: local authentication of the user, smart card handling, establishment of secure sessions, modules for access control and public key infrastructure [Morogan 2000].

1.6.2 Part 2: Certificate Management in Ad Hoc Networks

With the GSO platform in place, many security protocols and services become available in the ad-hoc network environment. The platform can handle all needs for encrypted communication, key exchange, signatures and creation of secure communication channels between devices. It also supports public key certificate usage and handling while connected to a classical network. In this section we discuss certificate usage and handling in an ad-hoc network scenario.

Public key certificates are widely used in network security systems as a proof of identity or proof of capabilities and rights. They are an important enabling technology for the implementation of several security services – authentication, non-repudiation and access control.

For usage in a global network, certificates need Public Key Infrastructure (PKI), a hierarchy of certification authorities that guarantee the validity of the certificates. The technology of certificates and PKI works within the environment of traditional wired networks. Here we analyze the problems in ad hoc wireless networks, and we propose some potential solutions. From the point of view of certificate usage, the most important limitation of wireless ad-hoc networks is the lack of permanent access to the global network.

Our presumption is that devices that form wireless ad-hoc networks also get connected to the Internet, but not necessarily at the same time. Therefore, most certificate management functions – e.g. request and receive new certificate – can be performed while being on-line. Verification of peer certificates however cannot be delayed or performed in advance – it must be performed on the spot, without regards to the availability of on-line access.

Analyzing the sequence of actions needed to establish a certificate’s validity, we find two steps that are dependent on on-line access: verification of the Issuer Certificate Authority (CA) and verification of the revocation status of the certificate.
Verification of the Issuer CA can be performed off-line if the certificate owner can provide a certificate chain, certifying the chain of CAs up to the top CA. Since this certificate chain can be cached by the certificate owner, this problem is easily solved.

Verification of revocation status is a more difficult problem to solve in an ad-hoc environment. All certificates have a certain validity period, but for various reasons they can be revoked before they expire (the keys can be compromised, or the role of the owner can change). When receiving a certificate, apart from verifying its correct signature and that it is within the validity period, one has also to verify that it has not been revoked under the time. This is done by downloading an up-to-date Certificate Revocation List (CRL) from the issuer of that certificate. These CRLs are updated at regular intervals. Devices can cache CRLs, but if the CRL validity time is short, the window of off-line usage will also be very limited.

To solve these issues, a CRL distribution scheme optimized for wireless ad-hoc networks is presented [Morogan 2003c]. The scheme is based on a combination of over-issued delta CRLs and indirect CRLs, and also on two special techniques: a distribution point for all CRLs in a domain, and peer-to-peer exchange of CRLs between users in an ad-hoc network.

Having a number of distribution points in a network that can all distribute all CRLs in the domain makes caching of CRLs by users feasible. The usage of over-issued delta CRLs contributes to a long window of off-line usage. Once off-line, clients with fresh CRLs can distribute these to other clients using the peer-to-peer CRL distribution protocol that we present.

Yet another technique for optimization of off-line usage window is to allow applications to specify their own requirements for the freshness of CRL information [Morogan 2003b]. In this way, it becomes possible to vary the timeliness requirements between applications and even within an application between different transactions, so that the whole system doesn’t need to stop working after a short time off-line just because one application has extreme timeliness requirements.

All these techniques together form a CRL distribution scheme that is much less reliant on centralized servers in the network, and that can provide improved off-line behavior, making it optimal for usage in ad-hoc wireless networks.

1.6.3 Part 3: Access Control for Documents in Ad-hoc Networks
With the GSO platform and the optimized certificate verification protocols, most of the security services available in classical networks become available in ad-
hoc wireless networks as well. These services include authentication, confidentiality, integrity and non-repudiation. In this section we investigate the issues related to the implementation of an access control system for documents in wireless ad-hoc networks.

Taking into consideration the particularities of this environment, we have set up a number of requirements for such a system:

- An access control system for wireless ad-hoc network should be self-sufficient, in the meaning that an access control decision can be taken without access to network resources.
- It has to protect documents at all times, even when stored on a device, and it should be enforced when a document is sent from one user to another.
- It has to support complex access control policies that can take into account factors such as the current location, domain of the user and of the peer.

To satisfy these requirements, we have designed an access-control system based on encapsulated documents and trusted decision engines [Morogan 2003d]. The idea is to separate the access decision functionality from enforcement and storage. Documents move freely through the network, being protected by cryptographic encapsulation techniques. Their content is not accessible by users in this form. They contain, also inaccessible to the users, a set of use conditions or access control policies. These policies can be specified in one of the several existing access control policy languages.

To gain access to the contents of a document, a user needs to employ the services of what we call a “Trusted Decision Engine” (TDE). TDEs are small servers that are available on trusted servers throughout the network or on trusted hardware on the client machine. Their role is to decrypt the access control policy from a document and, based on the extracted policy, on the general system policy and on the users’ credentials, take an access decision. If the decision is to allow the requested access, the key needed to decrypt the requested data is given to the user.

One advantage of this approach is that the TDEs can be made fairly compact, making it feasible to implement them on trusted hardware such as a smartcard or an i-button [Morogan 2003a]. Another advantage is that documents are protected through cryptographic encapsulation and therefore safe even in the case the device where they are stored is stolen, or if files are exchanged directly between users.

We further describe the extensions needed to the XML document standard to make them compatible with policy encapsulation and TDEs.
There are XML extensions that allow encryption and signature of the whole document or parts of it. Another extension exists for attaching access control policies to XML documents or parts of them. The scheme we propose provides a way to encapsulate these policies together with the content they protect, and make them accessible only to TDEs. In this way, a user can get access to the encrypted data of an XML document based on her credentials, by submitting a request to a local TDE.

1.7 Validation of Results

A fundamental aspect of designing security algorithms and protocols is analysing the results and making sure that they don’t have weaknesses or exploits that could be used by a potential attacker.

For the protocols described in this thesis, we have identified the attacks and threats that we considered relevant and analysed how the protocols counter them. Further the protocols have been released for review by the cryptographic community through publication.

Another important technique for increasing the security of a new design is to use as much as possible existing, proven technology. By basing the protocols presented here on existing cryptographic standards and technologies (encapsulation protocols [PKCS7], XML [Bray 2000], public key infrastructures [PKIX], smart cards [SCBasics]) the possibility of successful exploits is reduced, at the same time making an analysis of the novel aspects feasible.
Part 1:
Generic Security Objects - A Comprehensive Java Security System for Open Networks
This part is based on the Licentiate Thesis presented in November 2000 with the title “Generic Security Objects - A Comprehensive Java Security System for Open Networks”. It presents the design and implementation of a security system based on generic secure objects (GSO). The concept is to have a number of objects that provide advanced security functionality in a modular form. They are designed to be transparently interoperable, so that new functionality can be obtained by combining existing GSOs. The system is a platform on which higher level security services can be implemented.

The advantages of this approach are first of all modularity, ease of use, and flexibility. The system is modular, since only the required objects for a specific application are necessary – something that is important in systems with limited resources. The objects provide a high level of abstraction, making it easy to implement new security functionality and to use this functionality in general applications, therefore making application development quicker and less error-prone. Finally, new functionality can be created as needed by combining the basic objects, making for a flexible system.

The system was implemented on the Java platform. Apart from basic security functionality, it contains objects that provide high-level security functions, such as: local authentication of the user, smartcard handling, establishment of secure sessions, modules for access control and public key infrastructure.

The GSO system implementation provides a good platform for implementing and testing higher level protocols in wireless ad-hoc environments.
Chapter 2: Security Objects – Concepts and Principles

2.1 Introduction

The primary objective of this part of the thesis is to design and implement a security platform in open computer networks, distributed systems, mobile environments, and in heterogeneous application environments through use of generic secure objects. Such objects may provide multiple benefits through their enhanced and flexible functionality, mutual compatibility and scaling, rapid development facilities, and simplified evaluation/verification procedures. By generic secure objects we mean software objects with different security functionalities that through transparent methods can be used in various contexts. The research addresses three important aspects of generic secure objects:

- *description* of secure objects to cover the full functionality of a global and integrated security system;
- *design* of various constructors and methods for secure objects; and
- specification of a *scenario* for the usage of secure objects.

The research has created an initial theoretical basis and a practical platform for further design and development of secure objects and secure applications based on such objects. The main scope of this design is to provide flexibility and ease of use. It provides a different, higher level of abstraction when working with security mechanisms than other systems. The objects are designed to be intercompatible and provide very rich functionality – making it easy to combine them into other objects and create new protocols. Also, it is a goal of this work to provide a platform that is complete and flexible enough to be applied on a wide range of devices. This platform is the basis on which more advanced algorithms, introduced in part 2 and 3 of the thesis, are built.
2.2 Advantages of Security Objects

The concept of generic secure objects and their partial implementation performed in the course of this research clearly demonstrate the following advantages and benefits of such objects:

- Their security functionality, since many security functions may be implemented by simple instantiation of pre-established secure objects;
- Compatibility and scaling of secure applications created with generic secure objects, since the core technologies for such applications are uniform technical concepts, platforms, and components;
- Rapid development and broader applicability, since re-use of generic objects is suitable for fast development of secure applications for different user environments and for different technical platforms;
- Creation of Commercial Off-The Server environments, i.e. through remote invocations from specialized security servers, secure objects are available through Commercial Off-The Server arrangements;
- Evaluation and validation of secure objects is simplified, since users can use instances of secure, already evaluated and verified objects.

2.2.1 Security Functionality

Complex security services and protocols can be implemented by simply instantiating existing security objects. New services are also easy to implement, since objects can be combined and used as a basis for more complex objects. For this, it is necessary that objects share a common set of interfaces, so that values can be passed between objects or objects can directly work with other objects.

2.2.2 Compatibility and Scaling

It happens very often that systems that were originally designed for one task are used for something completely different and at a totally different scale after they are released [Foote 1996]. The Web standard is one example, originally designed to distribute scientific papers at CERN and now used to exchange all possible kinds of information over the whole world.

A system must be able to change and evolve in order to deal with the always new requirements that come up. An electronic commerce client, for example, could be developed to use password-based authentication. During its use, however, it may become necessary to upgrade the authentication system to use smart cards, and then later maybe to use biometrics authentication technology.

A system based on generic secure objects can be created so that it is flexible enough to deal with the changing requirements. Software based on it would be
easily scalable, since a component that does not scale well can be replaced by another one that does. It would also be flexible, adding new protocols and functions by just adding or replacing components.

2.2.3 Rapid development and broader applicability

As applications become more and more network dependent, the security risks they face are getting more serious. Security issues are also of prime importance to distributed systems and applications, something that is going to become very common with the emergence of wireless mobile computing and the usage of thin clients. The privacy of the individual is therefore becoming very exposed, especially since people start to depend more and more on computers.

Consequently, most applications should incorporate security features. There is an important difference between this and the current situation. Right now, the only applications that contain security features are the ones that are specially exposed to attacks, like for example programs concerned with transfer of banking data or with access to secret documents. These applications are developed by programmers specialized in computer security, who possess a great knowledge of the techniques and protocols involved. The rest of the programs and systems are very weakly protected, something that cannot be accepted on a large scale. However, even with the need of security becoming standard for all applications, it is not feasible to require that every developer has the deep knowledge required for developing a strong security application. This is one reason for using already built security objects.

Generic security objects can perform complex tasks, yet present a simple and straight-forward interface to the developer, who doesn't need any knowledge of what is going on inside. Making security functions simple to use not only makes development cheaper and less prone to errors, but it is also more likely to increase the number of applications to implement security. Applications that don't have a direct and obvious need for security functions usually don't have them. While this may not have a direct effect for one application, it lowers the general level of security of the whole system. For example, maybe it is not so important that the files written by a word processor are protected, but if all the files of a system are open to reading, information may be inferred from them. Also it would draw the attention to the few files that are not encrypted as being of much importance [Denning 1982, Morgenstern 1987, Smith 1988]. If on the other hand, the default behavior of all applications is to work securely, then the general level of security of the whole system increases. The idea here is that by making security functions easily accessible, it should be possible to make security the norm, instead of the exception, as it is now, and this would increase the privacy and the trust we can put in the computers, networks and information systems of tomorrow.
2.2.4 Commercial Off-The Server Systems

In the current model of software usage, an individual or a company has to own a certain program in order to use it. However, the situation often arises when some special software is needed for just one transaction or one data processing step. It is then not economically feasible to purchase the software and use it just once. In this kind of situation, it may be much more convenient to use software located remotely, and pay for that particular processing step instead of paying for the software itself.

Another situation where the need to use software located on servers may arise is in the usage of thin clients. These can be anything from mobile phones to low-power computers, and they may not have the computing power necessary for execution of the actual program. Even if they had it, the required module may not exist for that particular platform. Therefore, instead of porting one module to all possible platforms, different clients could communicate using a standard protocol (for example http) with a server that does all the processing.

In all these situations, off–the–server systems have clear advantages. Having a security system based on generic security objects makes the deployment of security functions at the remote servers easier, because the objects themselves are independent of each other and they can be used by a variety of different application programs.

2.2.5 Evaluation and Validation

Security software is more sensitive to programming errors than other kinds of software. With regular software, crashes and bugs are unpleasant and may have the effect of high maintenance costs. With security software, the costs can get much higher than just maintenance. A bank whose identification system ceased to function properly may in the worst case lose lots of money in forged transactions. Therefore, security software needs to be very thoroughly tested before being deployed in "real-life" applications with a high degree of sensitivity [DOD 1985, ITSEC 1991].

Of course, all software products are tested after they have been implemented and this is also the case with security software. This kind of testing reveals most of the errors. However, no kind of testing, no matter how thorough, can foresee every situation that the software could be put in, or every abuse or misuse that the software may be subjected to by either unknowing or malicious users. This is the reason why many software programs, after being tested for months during development, show up to have lots of errors only days after they have been released. After those bugs are fixed, others appear as the product is being used by many people during a long period.
It is therefore obvious that normal testing is not enough to achieve the high
degree of trust that is necessary for security software. One of the ways to solve
this problem in practice has proven to be the "test of time". While it is not
possible to test some software and prove that it is completely error–free, by using
some software for a long time and not finding any errors it is proven that the
probability of an error is very small.

Indeed, in computer security, unlike other domains of computer science, we tend
to trust most those parts that have been around longest. Speaking of algorithms
for example, we trust the DES encryption algorithm very much. Apart from the
theoretical strength of 56 bit encryption, the probability of some hidden door or
error that would make it easy to break is very slight. We do trust it because DES
has been around for many years and it has been tested in lots of real-life
applications. Many people have tried to break it or to find weaknesses in it, and
none have succeeded to find an attack that would go much faster than a brute-
force attack (or at least none that we know of) [Morris 1977, Lexan 1976, Davies
1982]. Therefore, DES has a high degree of reliability, something that is not
shared by other, newer symmetric encryption algorithms.

The same line of thought can be applied to software modules. A particular
implementation of DES that has been used many times is more trusted than a
new one. This is even more important for more complex protocols. Software
reuse becomes here imperative. Even more, there must be some guarantee that
the software is not being tampered with. For example, some library could be very
reliable, but if it has been modified for use in a program, its reliability can no
longer be guaranteed.

The best way to achieve programs with a high degree of security is therefore to
use existing and thoroughly tested security components. Software components
should be accessible through a public APIs and impossible to modify. So a
component is guaranteed to behave in the same way no matter what application
uses it. The reliability of a security component depends on how long it has been
in use, and therefore one can choose to build a system using components with a
certain known degree of security. Instead of implementing the security protocols
each time, and therefore being prone to new errors, through the use of old,
proven components, systems with better security can be created.

2.3 Related Research

A system that uses security objects to great extent is the Java platform. Being
designed as a portable and network-oriented programming language, it has a lot
of built-in security features [Fitzinger 1996]. Security in Java can be described in
two parts. First, there is the platform security, the part that is concerned with
running code on the local machine [Gong 1998]. This part verifies the code to be
run and continuously checks its execution, not allowing it to do anything that may be harmful for the host that is running it. The other part of the Java security architecture is dealing with cryptography and secure applications. It provides the mechanisms needed for applications to communicate securely over a network [CryptoSpec], like support for symmetric and asymmetric cryptography, certificate and key handling, access control [JCESpec], authentication and authorization [JAAS]. Java also includes support for Smart Cards [Guillou 1992, SCBasics, Vedder 1992] with the JavaCard platform [Zhiqun 2000, JavaCard]. The Java security platform has evolved a lot in the last few years, covering many of the weaknesses it originally had. The system is still though not comprehensive, leaving uncovered areas like certificate request and creation, standards-based encapsulation and security infrastructure components. Also, the objects in the Java platform are not designed as separate components that provide high-level security functionality, but as low-level implementations of the cryptography functions and services. Therefore, many of the advantages of the Generic Security Objects – separate validation, rapid development and deployment, interchangability – can not be obtained with the Java platform. This is of course a matter of different design goals – while the Java platform is a programming language that has to be able to be used for implementing any possible security application, GSO is a system that provides already-built objects with a higher level of abstraction.

Other work on secure objects include [Frincke 1996], which discusses the problems and techniques for developing secure objects. A system that is based on Modula-3 network objects, extending them with security functionality is presented in [van Doorn 1996].

2.4 Application Scenarios

We will use the following concept and structure of a comprehensive security architecture to design and demonstrate the use of secure objects:

Users are accessing and using various applications in environments with different security classification levels. For their personal interactions they may use PCs, mobile devices or any other personal tokens. Multi-user collaboration environments are established as local or remote interaction protocols, accessing local or remote application servers. Transactions in such environments may be exchanges of messages between two or more parties or transactions accessing and handling data, documents or other resources at remote servers. At remote servers, transactions may trigger back-end (database) operations or new actions on other remote servers. In order to provide secure and reliable operations, the complete collaborative environment must be organized within a secure infrastructure. That infrastructure must provide
− secure and reliable registration, certification, authentication, and authorization of all entities,
− secure message exchange and processing functions,
− secure remote operations, and
− management and administration functions of the global security system.

This scenario may also be considered as "generic", i.e. in principle it is applicable for any type of application and distributed data processing environment. It may be appropriately interpreted and adjusted to fit into the concept of secure applications ranging in scale from small, single station applications to any large scale, open network applications.

We will see how the above scenario looks like in a few examples, and from that we will describe how the comprehensive security system should look like.

![Diagram of a global security system]

**Figure 2.1**: The Structure of A Global Security System

Let’s suppose that a user wants to work with a client/server program. He has a small application client that talks with an application server over the network. The server can serve different categories of users, each with different access rights (for example, a user that does not authenticate himself may not be allowed to access sensitive data).

First, the user has to log in on the local machine and gain access to the correct private key and other sensitive personal data. After that, the client program identifies the user by sending its certificate to the server, which in its turn sends its certificate. Both parties now use their certification clients to verify the peer’s
A security protocol is then established between the client and the server. This has the purpose to ensure the authenticity of peers and also that the transmitted data is safe. Modules can exist for different kinds of protocols, like for example protocols for strong authentication, virtual networking, secure data transmission, contract signing or electronic commerce.

The modules involved in this example are shown in Figure 2.1.

We take now the example of a security system designed for being used in a local area network. A user needs some service that is provided by the Application Server. Since not all users have access to that service, or even if they have, it is somehow restricted, the server needs to know who the user is. Since the system is local, the server can have a registration database with all the users. Each user is assigned a certificate to show its identity.

We see that here we can use the same modules that we used in the previous system, the one for a global network, only that here we only need one CA (Figure 2.2). The user logs in on the local machine and gains access to his personal authentication data. Second, the user and the server exchange certificates which they verify using the Certification Client module. After identifying each other, they may run a strong authentication protocol.

Figure 2.2 : Local Area Security System
We are now going to analyze generic security modules placed on the local machine (see Figure 2.3).

The central part of the whole system is the Login module. Its main function, after correct login, is to provide access to the private key of the user to other objects that need it. Whether the actual private key is available outside of the Login module or not, is a design question. One way to use private key is that instead of getting it out from the Login module, programs give data they want to process to the Login module which then processes them and returns the result. In this way, the key never leaves the Login module, and if the Login module is implemented using smart cards, this separation becomes much stronger.

Registration module is responsible for all functions handling user/server registration data. It creates initial registration object, updates its data, if needed, stores registration data in various devices (if they are used) and distributes registration data to other (local and remote) modules, when those data are needed. One of the most important "peers" of the Registration module is Certification module.

The Certification module handles all functions related to certificates. It takes care of creating a new certificate request, if the user needs a new certificate. It also submits certification request to the CA server and receives back the new certificate. When another certificate has to be verified, not only the signature is checked, but also the chain of certificates, that guarantees that it has not been issued by anybody else but the local Certification Authority. An important function of the Certification module is to verify the Certificate Revocation Lists (CRL) and see if the certificate or if one of the certificates in the chain have not been revoked.

After successful login, if registration or certification modules are not invoked, the user is ready to initiate secure session with some remote server. Security session module supports different client/server secure protocols: three-way strong authentication, SSL protocol, secure electronic commerce protocols, and so on.

Finally, Security Administration module is dealing with all the local settings, access rights, and also certificates, protocols to use, maybe delegation of authority, if such protocols exist in the Secure Session module.
In Figure 2.4 all components of the system are shown. The different applications, shown in white boxes, run on top of them. The security modules are shown shaded, suggesting that all data that they handle are protected. Data is therefore protected not only during transmission, but at all times, even during storage on a server. The Message module takes care of translation between messages encrypted for storage on a server (protected with the public key of the server) and messages encrypted to be sent to clients (protected with the public key of the client).

Figure 2.3 : Local Security Modules

Figure 2.4 : Complete Client/Server Security System
A system based on this architecture is able to provide the functionality needed for the scenario presented at the beginning of this section. In subsequent chapters we will describe in detail the architecture and security objects involved in the different modules of the system.
Chapter 3:
Specification of Generic Security Objects

A global security system is comprised of generic secure objects, security protocols and components of the security architecture. Security objects are components which are used to provide various security features and properties of the system. Objects perform that by the data they contain (called security elements), by actions performed on individual objects or performed by individual objects over other objects (called security methods) or by interactions or combinations of two or more objects (called security protocols).

An example of the object which provides security by the data it contains is DistinguishedName. The elements of this object are used in many situations where unique identification of entities is needed. An example of a security object which performs actions on other objects and thus provides security feature is SymmetricKey. Its actions are encryption and decryption of other objects (messages). SymmetricKey is an example of an object which is self-sufficient to produce some security effect on messages. Security actions using this type of objects will be called security methods provided by objects.

However, some objects can not produce any effect unless they are combined with other secure objects. An example of such an object is Hash. That object applied to data message produces message authentication code, but that is not enough by itself. In order to provide data integrity, authentication code must always be further protected by encrypt or decrypt method of either PrivateKey or PublicKey objects, creating finally Signature object or EncapsulatedData object. This combination of two objects and their methods to produce specific security effect will be called security protocol. The given example shows signature protocol and encapsulation protocol. Other examples of security protocols may be certification protocol, strong authentication protocol, etc.

Combinations of secure objects, their methods and protocols may be used to create a global security system, where the desired security effects are achieved by
- *Instantiation* of individual generic security objects,
- Usage of various *data* and/or application of various *methods* provided by instantiated objects, and/or
- Combination of various objects, their data and methods with other objects, i.e. *security protocols*.

But, collection of security objects, with their data, methods and protocols is not sufficient to establish a complete security system. What is additionally needed are various, in this thesis called, *security utility* objects. Such objects represent additional aspects needed for global system security, representing security policies, time or authorization constraints, trust schemes and levels, etc.

So generic security objects can be classified into security resource objects and security utility objects. In this research security resource objects are understood as objects existing in the system/network which either perform some action or some action is performed on them. Security utility objects are additional objects used indirectly to specify, establish, control, and enforce security in a given environment. This thesis will consider in detail only security resource objects, not utility objects.

The high level hierarchical structure of secure objects may be the following:

**Security Objects**

**Security Resources**

- **Permanent** – stored in the system (exist beyond sessions)
- **Active** – they do something with other (passive) resources

**People**

- Users – using resources in the network
- Administrators – setting parameters for operation
- Managers – deciding on policies, profiles, authorizations

**Software modules**

- Programs – static (in files or libraries)
- Applets – dynamic (single source, single destination)
- Agents – dynamic (single source, multiple destinations)

**Hardware modules**

- Identification devices – challenge/response tokens
- Storage devices – read/only storage devices, like memory smart cards
- Crypto devices – crypto smart cards
- Active devices – Java cards, iButton

**Passive** – used by active resources

- Data objects – objects being processed
- Signature
Encapsulated data – message, file record, HTML page, E-mail letter

Simple files
Data base segments
Parameter objects – objects needed for secure data processing
  Identity – friendly name
  Password – secret authentication parameter
  DistinguishedName
  Registration data – everything else other then DN attributes
  Symmetric cryptographic key
  Asymmetric cryptographic keys
  Certificate
  CRL
Capability objects
  Tickets – specific access capability objects
  Authorization data – specific functions capability objects
  Digital money – general capability objects
  Capability identifications – indirect capabilities

Temporary – exist only during individual sessions or messages
  Password
  Random numbers
  Hash
  Challenge/response parameters
  Tickets/tokens
  Symmetric crypto keys

Utilities
  Consistency Schemes
    Time
    Non-repudiation
  Authentication Schemes
    Policy
    Trust
  Authorization Schemes
    Authorizations
  Recovery schemes
    Archive
    Backup
We are now going to describe the individual objects of the global security system. A complete listing and description of all objects, including relevant methods, can be found in the Appendix.

### 3.1 Identification and Authentication Objects

Objects in this group are used to store data related to an entity and to perform operations necessary for authenticating the entity. There are five objects in this group:

- DistinguishedName
- UserIdentity
- AuthenticationObject
- Login
- UserRegistration

**DistinguishedName** object is used to store the DistinguishedName of the user, as specified by the X.500 standard [X500]. A user may have several DistinguishedName:s, with different attributes (for example one from work and another one from his bank). All DistinguishedName:s of a user are stored in the registration file.

**UserIdentity** is a wrapper for several objects belonging to the user: user name (user-friendly name), one or several DistinguishedName objects and the login key.

![UserIdentity Diagram](image.png)

**Figure 3.1 : UserIdentity**

The **UserIdentity** object represents a user that has logged into the system. It contains both a user-friendly name of the user (the login name), the DistinguishedName (one or several), and the login key that is used to protect the user registration data and private keys.
The AuthenticationObject contains data needed to verify the local login. It also contains data about the policy of the local login, like for example minimum password length or the date after which the password must be changed. It returns a symmetric key that can be used to protect the PrivateKey:s of the user and other sensitive data.

When a user logs in locally, he or she has to supply an identification and some form of verification. Once the user has entered the login data, the login system uses the AuthenticationObject that belongs to that particular user to verify the authenticity of the user. The result of this verification is returned to the Login object.

The form of authentication can be classical name/password control, where the user identifies herself with a nickname and proves her identity with a password known only to the user. This method is the one that is most widely used right now, even if it has the weakness that passwords can be easy to guess [Morris 1979, Abadi 1997]. In the case of using smart cards, the identification is the smart card itself, while the authentication information is the PIN code used to unlock the card. Other more advanced implementations can be the use of biometric techniques, like face recognition and/or fingerprints to identify and authenticate the user [Huopio 1998, Davies 1994].

For any of these implementations, the authentication data obtained in the login process, whether it is a password, a PIN code or biometric parameters, is passed to the AuthenticationObject where it is verified. The AuthenticationObject stores the information needed to verify the login in a file. This file is sensitive and therefore should be protected. In case of smart cards, for example, the authentication file is stored inside the card, and is not accessible from the outside [PKCS15]. In other systems, if the authentication file has to be stored in an unprotected area, it can use one-way functions so that it can verify the authenticity of a request, without containing the secret data itself.

The AuthenticationObject controls the access to the rest of the user’s sensitive data, like the user’s PrivateKey:s for example. How this is done depends on the implementation. In a trusted hardware device, like a smart card, the AuthenticationObject gives access rights to the area where the sensitive data is stored. These access rights are enforced by the device. If all data is stored on a computer, the AuthenticationObject contains a symmetric key (which we call login key) that is used by the system to encrypt all other sensitive user data. This login key itself is protected by encryption, using the user authenticity information as a key. After a successful login, the login key is decrypted and made available, so that the user’s private keys can be decrypted and used. The advantage of using this approach instead of encrypting the user private keys with
the user password is twofold. For the first, it is more flexible, allowing for
authentication systems that have other means of authentication then a password.
For the second, if the password is changed, there is no need to re-encrypt all
private keys and sensitive data that had been encrypted with it. The only thing
that needs to be done is to decrypt the login key with the old password and
encrypt it anew with the new one.

![Usage of AuthenticationObject](image)

**Figure 3.2**: Usage of AuthenticationObject

The constraints for a password (i.e. minimum length, how often it is to be changed) are specified in the policy file of the system. This file contains sensitive information and should be protected, either by storing it in a safe place (on a smart card for example), by signing it, or both.

**Login** object performs the login. It gets the identification and verification information from the user and verifies them against the **AuthenticationObject**. If ok, then sets the flag “loginSuccessful” and waits for requests. It can provide the following objects: **UserIdentification**, **UserRegistration**, and a symmetric key, called **loginKey**, obtained from the **AuthenticationObject**, which can be used to unprotect security sensitive items, like **PrivateKey:s**.

When logging into a computer, the user supplies her identification and authenticity information (whether it is a password, a PIN code or a fingerprint) to the **Login** object. In its turn, the **Login** object takes the authenticity information supplied by the user and sends it to the **AuthenticationObject** for verification.

If the verification does not succeed, the **Login** object prompts the user for another try. If the **AuthenticationObject** triggers the error **OldPasswordException**, the **Login** object asks the user to give a new
password and then tries to change it by calling the changePassword method in AuthenticationObject.

![Diagram of Login object]

**Figure 3.3 : Login object**

If the verification succeeds, Login reads the login key from the AuthenticationObject. It then creates a UserRegistration and a UserIdentity object for the user. Other objects can then request the UserRegistration or UserIdentity objects from Login.

A UserRegistration object is created by Login after successful local authentication. It is initialized with the user name and fetches all the user information from the user registration file. This file also contains the DistinguishedName or names of the user.

UserRegistration object contains all the data about the user that might be necessary. For example, it may contain the address of the user and other registration data which are not attributes of the DistinguishedName.

UserRegistration also contains the DistinguishedName:s of the user. One of them is marked as the default DistinguishedName and is used when no other one is specifically requested.

All this is stored in the user registration file. This file may be stored on the local disk, a network server, or a smart card, depending on the implementation. Some parts of it need to be protected during storage, so they are encrypted with the login key. The actual structure and content of the registration file depends on the particular implementation, and a special subclass of the UserRegistration needs to be created for it.
3.2 Cryptographic Key Objects

This group consists of four objects used to store keys and perform cryptographic operations. They are:

- AsymmetricKeyPair
- PrivateKey
- PublicKey
- SymmetricKey

An AsymmetricKeyPair object consists of a PublicKey and its corresponding PrivateKey.

PublicKey contains an asymmetric public key and is used to encrypt and decrypt data, and to verify signatures. PrivateKey contains an asymmetric private key. Its usage is to encrypt, decrypt and sign data. A PrivateKey object can be retrieved from the system database by specifying its corresponding PublicKey and the UserIdentity object of the user. UserIdentity object is used to open the access to the key. If the key is stored on the local disk, then the login key from the UserIdentity object is used to decrypt it. If the key is stored inside a secure hardware device, the UserIdentity is needed to gain access rights to it.

SymmetricKey object contains a key used for encrypting and decrypting data with a symmetric algorithm. The main advantage of a symmetric algorithm compared to an asymmetric one is that symmetric encryption and decryption are generally an order of magnitude faster on comparable implementations [Eberle 1993, Brickell 1990]. Therefore, bulk data is usually encrypted with a symmetric algorithm. The object also contains methods for encrypting the key itself prior to storage or transport over a network.

3.3 Certification Objects

This group consists of the following five objects, all related to certificates and certification infrastructure.

- Certificate
- CertificateChain
- CertificationRequest
- CertificationClient
- CRL
A Certificate contains the certificate of a user, server or program, according to the X.509 standard [X509]. It is created by a Certification Authority (CA) server, subsequent to a CertificationRequest object sent from a CertificationClient. It contains, among other data, a PublicKey, a DistinguishedName associated with that public key and the signature of the Certification Authority (CA) that certifies this association. It may also contain the Certificate or the CertificateChain of that CA and other data related to the user, which is not contained in the DistinguishedName.

A CertificateChain object consists of several Certificate:s that certify each other. It is used to prove that a certain Certificate is certified by CA:s all the way to the top of the hierarchy.

![Certificate chain and the certification hierarchy](image)

**Figure 3.4**: Certificate chain and the certification hierarchy

To obtain a new Certificate, a client has to create a CertificateRequest and send it to the CA. The CA can then verify the request and create a new certificate for the client. A CertificateRequest has to contain the PublicKey and the DistinguishedName of the client according to the PKCS#10 standard [PKCS10].

The CertificationClient is the part of the user local system that takes care of all communications with the certification infrastructure. It can send a certificate request and receive the response from the CA, or it can verify a Certificate object, checking its credentials and also checking that it is not revoked in a CRL [RFC2459]. To create a new Certificate, the CertificationClient takes all data necessary and creates a CertificateRequest object, which is then sent to the default CA. The CA can then be asked if the Certificate has been issued or not. If the CA has
issued the Certificate, it is fetched and stored in the system database by the CertificationClient.

The name and address of the default CA, as well as other parameters necessary to the CertificationClient are stored in a file, called Certification Policy file. The CertificationClient needs also to store the certificate of a known and trusted Certification Authority. This certificate is also stored in a file on the disk or inside a smart card.

CRL stands for Certificate Revocation List and it contains the numbers of all certificates that have been revoked before their expiration date together with a reason for their revocation [RFC2459]. CRL:s are received by the CertificationClient from the CA:s.

When receiving a certificate, the first thing to do is to verify it, that is to check that the signature that certifies the association between the PublicKey and the DistinguishedName is correct. For this, the Certificate of the CA is used. That Certificate must be verified in its turn, and so on until the certificate of a known and trusted authority is found. This verification can be done in two ways. The first way is to verify only the validity of the signature, without verifying the certificate of the signing authority. This can be done for example when the signing authority is known to the receiver. The method used for this is
bool verify()

and it returns true in case the signature is correct. The other way of verifying the
certificate is to submit it to the CertificationClient object, which in
its turn verifies each certificate in the chain until it finds a certificate that it trusts,
and also checks whether a certificate is not revoked in a Certificate Revocation
List (CRL).

3.4 Secure Session Objects

These are two objects – EncapsulatedObject and SecureSession – used to
implement higher level cryptographic protocols related to data transmission and
storage.

An EncapsulatedObject contains a normal Java Object, which it encrypts
and signs with the asymmetric keys that are supplied to it. The
EncapsulatedObject can be transformed into a byte array that can be sent
over the network and then used to create a new EncapsulatedObject. The
object that is contained in the EncapsulatedObject has to be serializable. By
supplying several PublicKey’s, the resulting EncapsulatedObject can be
made readable by several recipients. In the same way, by supplying several
PrivateKey’s, multiple signatures can be added.

To produce an encoded byte array, the method getProtected is called. Then
the Object inside the EncapsulatedObject is serialized, encrypted with a
symmetric key which is automatically generated and possibly signed with the
private key of the sender. The symmetric key used to encrypt the serialized
Object is in its turn encrypted with the public key of the receiver and the result
is encoded in a byte array, according to the standard specified by the
encoding_id. Before encrypting the object, other data may be added to the
EncapsulatedObject. For example, several receivers can be added, making
the object readable for all of them. This is done by making several copies of the
SymmetricKey used to encrypt the bulk data, and encrypting these copies, each
with the public key of a new receiver. These separate encrypted symmetric keys
will then be added to the package, making it possible for all receivers to obtain
the original symmetric key and decrypt the contents of the message. In the same
way, several signatures can be added to an EncapsulatedObject. This can be
useful when a message with multiple senders is sent and each sender needs to
sign it.
The format of the resulting byte array can be specified when calling `getProtected`, making it possible to choose the standard to be used (for example PKCS#7 [PKCS#7]). Once the protected byte array has been created, it can be sent over the network or stored on a local disk. A new `EncapsulatedObject` can later be created from it. The `PrivateKey` of the user, necessary to decrypt the message, is fetched automatically from the database.

The `SecureSession` object opens a secure communications session with another entity on the network. Its main function is to perform a strong authentication protocol between the two entities and then open a secure channel. The object is initialized, then it produces messages that are to be sent between the parties. The messages received are at their turn given to the object, which can then check if the process is going on correctly, and if it is, create the next message.

`SecureSession` needs the following items: the `Certificate` of the local user, the `Certificate` of the peer, and the `PrivateKey` of the local user.

The result of the authentication process, if it has been successful, is a `SymmetricKey` object (session key) that can be used for further secure communication between the parties.

### 3.5 Access Control and Authorization Objects

There are two objects designed for usage in access control and authorization systems: `Capability` and `AttributeCertificate`.

A `Capability` represents an action that is allowed to its owner. It can be for example a movie ticket, granting the entrance of its owner to the theater, or it may allow access to a system resource. The `Capability` itself does not contain...
any information about its owner, neither is it protected or authenticated in any way.

A Capability consists mainly of a name that identifies it and a description. It may be subclassed to contain other data as well, depending on its intended usage. For example, the movie ticket could be called just “Movie Ticket”, its description may be “Virtual piece of paper granting entrance to the theater” and it would contain, as additional parameters, the starting time of the movie, its name and the number of the seat.

An AttributeCertificate represents an authenticated Capability. It contains a Capability object that is signed by its issuer. It may also contain the name of its owner, if that is needed (some authorizations may be anonymous) and the DistinguishedName of the issuer.

![Figure 3.7: Structure of the AttributeCertificate object](image)

3.6 Smart Card Objects

This is a set of objects designed to facilitate the usage of smart cards in secure applications. The objects are:

- SmartCard
- Terminal
- SCSession
- SCAplication

The SmartCard object provide methods for activating, initializing, and managing smart card contents.

The Terminal object methods provide smart card reader devices management and access to the smart cards functions.
SCSession provides methods to manage applications or objects on a smart card via secure or regular smart card sessions. Session is a logical combination (collection) of smart card applications that a user, who opened the smart card, may use. Sessions may be dynamically opened and closed, i.e. the user may activate different combinations of smart card applications, depending on user needs and the IT environment.

The methods of SCAplication object provide the possibility to create, update, delete, and activate smart card applications.
Chapter 4: Protocols and Usage of Security Objects

In Chapter 2 we described a scenario for the usage of the Security Objects. In this chapter we are going to show how the protocols used to complete those scenarios work.

The first thing a user has to do is to authenticate herself locally to the local machine by going through the login process. Once this is done, she can go on and use different services and protocols supported by the system.

We are going to go through the following actions:

- login on the local machine
- create an encapsulated object
- run a strong authentication protocol with a remote entity on the network and create a secure channel after the authentication has succeeded.

We will go through each of these actions and see how they are done in two types of systems: system that do contain a secure, trusted hardware device, and systems that do not. We will call the systems without a trusted device “classical systems” and the others “smart card systems”.

4.1 Local Login

We will first see how things work in the case of a classical system.

First of all the user has to login to the local machine. To do that, she has to supply her identification and verification information to the Login object. Through some GUI, she writes her login name and her password. The Login object takes these and passes them to the AuthenticationObject. This object reads the authentication file corresponding to that particular user name. In that file there is some known data, like for example the user name, encrypted with the user password. Note that the password itself is not stored in this file, because the
file is stored in an unsafe environment (a computer disk). If the user name encrypted with the supplied user password gives the same result as the one written in the file, then the login is accepted. If the results do not match, the Login object is told that the login was not successful, and it asks the user for a new password. Otherwise, the process continues.

In the authentication file belonging to the user there is also stored a SymmetricKey, also encrypted with the password of the user. This key is the one that is used to protect all other confidential data that the user has, like for example her PrivateKey:s. The key is now decrypted using supplied password, and after that the AuthenticationObject responds to the Login object that the login has been successful.

Login reads the SymmetricKey from the AuthenticationObject. This key is called the login key. Login now creates a UserRegistration object with the data from the user’s registration file. From the same file, Login reads the DistinguishedName (one or several) of the user. It then creates a UserIdentity object which contains the user login name, the login key and the DistinguishedNames.

![Diagram of the login process]

**Figure 4.1:** The Login process

The login process is now finished. The Login object can now receive requests for the UserIdentity or the UserRegistration of the user.
In a smart card based system, the identity of the user is already established, since smart cards are personal. What has to be established is if the user really is the owner of the card. For this, a PIN code is entered. This PIN code is passed by the Login object to the AuthenticationObject, which contains the correct PIN code. This is possible here because on a smart card, the AuthenticationObject is protected. The two PIN codes are compared. If they are not the same, the Login object is signaled and it prompts the user for another try. After three failed attempts for login the card is locked. If the two PIN codes are the same, then the login is successful and the access to the PrivateKey:s and all other data of the user stored on the smart card is opened. However, note that it is now possible for programs to call methods on objects like the PrivateKey:s, but not to read their internal data.

![Figure 4.2: Login on a SmartCard](image)

The Login object gets the login key from the AuthenticationObject and creates a UserIdentity and a UserRegistration object. All user registration data is read from where it is stored, either on the card or on the disk (it would be best to store all registration data inside the smart card; however, due to memory limitations, this may not always be possible).

### 4.2 Creating a Secure Session

To communicate securely, entities on a network have to prove their identities to each other. This can be done using the SecureSession object.

First of all, in order to start communicating, the entities have to establish contact and identify themselves, for example by exchanging certificates. When the Certificate:s of the peers are received and verified by the CertificationClient, the parties need to have some proof that the entity
which sent them the Certificate really is the owner of that Certificate. Therefore, a three-step strong authentication protocol is performed. During this protocol, the parties prove to each other that they possess the secret PrivateKey corresponding to the certificate they sent.

The protocol described here is designed as a mechanism conform with the SASL (Simple Authentication and Security Layer) specification [RFC2222].

The first step of the strong authentication protocol is that one of the parties, let us call it A, sends a packet to the other party, which we will now call B. This packet contains a SymmetricKey generated by A, which is to be used later for creating a common communication key. Apart from the SymmetricKey, there is also a random number that is called “challenge”. The role of this number is to prevent replay attacks. Replay attacks are when somebody is recording the packets sent from B and using them to complete an authentication with A at a later moment. The whole packet is signed by A and encrypted for B.

On the other side, B produces a similar package, containing B’s SymmetricKey and challenge.

When receiving the package from A, B decrypts it with her PrivateKey and verifies the signature. If the verification is correct, B extracts A’s SymmetricKey and combines it with her own. The obtained SymmetricKey is the session key to be used for further communication. A does the same thing with the package it received from B. Both parts now have the same SymmetricKey for communication, but the authentication process is not finished yet: the challenge must be sent back. Therefore, a new package is created on both sides. A takes the challenge received from B, encrypts it with B’s PublicKey, then adds her own signature. B does the same with the challenge received from A, and then the packages are exchanged. Now both parts can verify that the peers could decrypt the challenge, so they must be in the possession of the correct PrivateKey:s.

This process is executed by the object SecureSession. This object is initialized with the Certificate:s of the local user, the remote entity, as well as the UserIDentity.

Once started, the SecureSession produces messages by calls to the method getNext(). What actually happens inside is that an EncapsulatedObject is created for each of these messages. The EncapsulatedObject is then transformed to a byte array with a call to getProtected(), and a byte array is returned from the SecureSession object by getNext().
Once a new response has been received, it is passed to SecureSession with the method receive(message). SecureSession processes this message. If the verification fails, an AuthenticationException is triggered. Otherwise, it is ready to produce the next message. When the protocol is completed, the common SymmetricKey is available from the object SecureSession, so A and B now share an encrypted “channel” over the network.

4.3 Creating an Encapsulated Object

An EncapsulatedObject is created to protect a normal Java Object for transport or storage. At initialization, or through subsequent calls, the EncapsulatedObject is passed the object to be encapsulated, the Certificate and UserIdentity of the sender, and the Certificate of the receiver.

All actual processing of data is done when the getProtected() method is called. First, the object to be protected is serialized and transformed into a byte array. This byte array is now signed with the PrivateKey of the sender, then everything is encrypted with the PublicKey of the receiver.
The **PrivateKey** of the sender may not have been given as a parameter to the **EncapsulatedObject** from the start (though there is a constructor that allows the **PrivateKey** to be specified directly).

In this case, the **PrivateKey** of the sender has to be fetched from the system key database. From the certificate of the user, the **PublicKey** is taken, and it is used to retrieve its corresponding **PrivateKey** from the database. That key is protected, so the **UserIdentity** is used to open access to it. In a classical system, the login key from **UserIdentity** is used to decrypt the **PrivateKey**. In the case of a smart card system, the **UserIdentity** opens the access to the **PrivateKey** stored inside the card.

Now the byte array could be signed by being encrypted with the **PrivateKey** of the sender. However, since asymmetric encryption is very slow, it would be too expensive to encrypt the whole byte array.

The approach used instead is to first produce a much smaller, fixed length byte array from the original one. This is called hashing. The hash vector is then encrypted with the **PrivateKey** of the sender. Since its size is very small compared to a typical data array, the operation goes very fast.

All this is done automatically by the **PrivateKey** object when it is asked to sign data. The output from the **PrivateKey** `encrypt(data)` method is the encrypted hash vector. Verification goes in the following way: the hash is decrypted with the **PublicKey** of its sender, then compared to a newly produced hash for the same data. If the hashes match, the signature is correct.

Now that the signature has been produced, it is packed together with the original byte array, following some standard encoding.

The next step is to encrypt both the serialized object and its signature. This is done by taking the byte array that contains them encoded together and encrypting it with the **PublicKey** of the recipient.

Now here we have the same problem that we had with the signature: asymmetrical encryption is very resource-expensive, and encrypting all data with it is not feasible.

Therefore, what is done is that a **SymmetricKey** is randomly generated inside the **EncapsulationObject** and used to encrypt the byte array. The result of this is the byte array, containing the serialized object and its signature, now encrypted with the randomly generated **SymmetricKey**. Since this key is needed
for decryption, it must be sent along with the data. For this it is encrypted with the PublicKey of the receiver. In this way, instead of encrypting asymmetrically all the data that needs to be sent, we are just encrypting a small SymmetricKey. This encrypted SymmetricKey is now added to the byte array encrypted with it. The Certificate of the signer can also be added here.

All these items are encoded together in the same byte array. This is done according to some encoding standard, default is PKCS#7 [PKCS7]. The structure of a PKCS#7 encoded message for signed data is the following:

Other encoding standards can be used as well, by specifying them in the call

getProtected(String encoding_id);

The main difference between creating an EncapsulatedObject using a classical system and using a smart card is that with the smart card, all encryption is done inside, so the keys - even the randomly generated SymmetricKey - cannot be read from the memory of the computer.
The byte array that is the encapsulated object can now be sent to its receiver. There it will be decrypted and the signature verified, by creating another EncapsulatedObject with it. In the following example, an object is created from a received byte array. The Certificate of the user and her UserIdentity are also given to the constructor, so that the PrivateKey of the user can be found.

EncapsulatedObject received_obj = new EncapsulatedObject(received_data, user_cert, user_id);

When instantiated like this, the EncapsulatedObject first retrieves from the system database the PrivateKey that corresponds to the user Certificate. Then it tries to decrypt the SymmetricKey from the received data. If this is successful, it tries to use this SymmetricKey to decrypt the rest of the data. If any of these operations are not successful, a null object is returned. If the operations are successful, the EncapsulatedObject now contains the decrypted byte array that is the serialization of the original object, as well as the signature for this array. The signature is now verified, using for that the public key from the certificate of the sender. Again, if the signature is not correct, a null object is returned. If even this step passes without problems, a new object is created from the serialized byte array. So now the EncapsulatedObject has the same contents as it had when it was created for encrypting data: it contains a Java Object that is the message, and it contains the Certificates of the local user and of the peer; these could be used to get the PrivateKey of the user and
sign the message, respectively to encrypt the message. The object that is the message of the EncapsulatedObject can be obtained by calling the method

Object getContents();

The certificate of the sender can be verified by first getting it out from the EncapsulatedObject and then submitting it to the CertificationClient for verification:

Certificate peer_cert = received_obj.getSigner();
bool correct = CertificateClient.verify(peer_cert);

4.4 Using Access Control

After successfully establishing a secure session, the use of access control becomes necessary to decide what the peer entity is allowed to do on the server. One way to implement access control is by having an Access Control List (ACL) on every server, where it is specified for each client what rights it has. Since this can become difficult to implement in a network with many clients and many servers, we have chosen to use a system based on capabilities and authorizations. One (or several) central servers contain the ACL:s for the whole system. The client needs to first go to one of these servers (named Access Control Authorities – ACA:s) and get an AttributeCertificate for the action it plans to perform, then it can go to any other server and use the AttributeCertificate [Johnston 1998].

![Access Control Diagram](image_url)

Figure 4.6: Access Control
First, the client will go to the ACA and, after establishing a SecureSession with it, it submits a request for AttributeCertificate for the specific action. This request consists of a Capability object for that action, and the client’s Certificate. The ACA checks the Access Control List to see if the client is permitted the specific action. If it is, then it creates a new AttributeCertificate object from the Capability it received, then sends it back.

The client can now open a connection to the Application Server it needs, then send the request for that action, together with the AttributeCertificate object it received from the ACA. The Application Server checks the signature of the ACA, then checks that the Certificate object present inside the AttributeCertificate is the same as the Certificate that has been obtained during the initialization of the SecureSession. If everything is correct, the client is allowed to perform the requested action.

4.5 Evaluation of the System

In Chapter 2 we described a scenario for the usage of Security Objects. The analysis of this scenario lead to a structure of the system, made of several security modules. We will now show how the Security Objects that are described in Chapter 3 and Chapter 4 can be used to implement the security modules used in that scenario.

Login module: formed by the objects Login and AuthenticationObject. These objects accept the authentication information from the user and, if it is correct, open the access to the private key of the user and other protected items.

Registration module: formed mainly by the UserRegistration object, which takes care of the user registration data, with the AuthenticationObject taking care of introduction of new users to the system and managing their passwords.

Certification module: implemented with the help of the objects CertificationClient, Certificate, CertificateChain, CertificateRequest and CRL, which together handle the process of verifying certificates and creating new ones.

Security Session module: formed by the SecureSession object, which here performs the protocol of remote authentication.
Security Administration module: not made by any of the objects described here, but its functionality is obtained by modifying the files that control the behavior of the objects dealing with security administration:

- authentication file used by the `AuthenticationObject`
- registration file used by `UserRegistration`
- policy file used by `AuthenticationObject`
- policy file used by `CertificationClient`

There is also a `Message` module that can be implemented by using the `EncapsulatedObject`, encrypting objects for storage and decrypting them back again.

We are now to show an example of the usage of Security Objects by going through one of the scenarios described in Chapter 2.

In this scenario, the user wants to work with a client/server program. He has a small application client that talks with an application server over the network. To use this system, the user has first to log in on the local machine and gain access to the crypto keys. For this, she gives her name and password to the `Login` object, through a GUI. The `Login` object controls the correctness of the authentication with the help of the `AuthenticationObject`. If the username and the password are correct, `Login` will receive the access to the `PrivateKey` of the user. Now the user is logged in to the local machine.

The client program running on the user’s machine identifies now the user to the server by sending the user’s certificate. The server sends its own certificate back, and now both parts have each other’s certificates. They verify them using the `CertificationClient` object.

Now that both parties have received the identity of the other, what they need to do is to authenticate each other. For this, a `SecureSession` object is instantiated on both parts. Messages are obtained from it and sent over the network to the peer. Once there they are given to the local `SecureSession` which verifies them, and if they are accepted, the next message is produced. When the strong authentication protocol is completed, the user and the server have a channel to use to communicate securely. Now the user can start using her application client to connect to the server through this secure channel, and the server can give the user the suitable access rights, based on the user’s identity.

With this, the scenario is finished.
4.6 Fulfillment of the Requirements

The requirements for the security infrastructure were, in Chapter 2:

− secure and reliable registration, certification, authentication, and authorization of all entities,
− secure message exchange and processing functions,
− secure remote operations, and
− management and administration functions of the global security system.

The Security Objects provide the following functions:

− secure and reliable registration - through the AuthenticationObject and UserRegistration objects.
− certification - through the CertificationClient.
− authentication - through the SecureSession.
− authorization - through the AttributeCertificate and Capability objects.
− secure message exchange and processing, achieved by using the EncapsulatedObject.
− secure remote operations through the SecureSession.

Management and administration functions are not present in the system, as they are more of an application than part of the infrastructure itself.
Part 2:
Certificates in Ad Hoc Wireless Networks

The focus of this part is to analyze and suggest optimizations related to the usage of public key certificates in ad-hoc networks.

Public key certificates are widely used in network security systems as a proof of identity or proof of capabilities and rights. They are an important enabling technology for the implementation of several security services – authentication, non-repudiation and access control.

For usage in a global network, certificates need the backing of a Public Key Infrastructure (PKI), a hierarchy of certification authorities that guarantee the validity of the certificates. The technology of certificates and PKI has been developed and designed to work within the environment of traditional wired networks. In this part of the thesis we analyze the problems that appear in the different environment of ad hoc wireless networks, and we outline some potential solutions.

Verification of revocation status is a difficult problem to solve without direct on-line access. All certificates have a certain validity period, but for various reasons they can be revoked before they expire (the keys can be compromised, or the role of the owner can change). When receiving a certificate, apart from verifying its correct signature and that it is within the validity period, one has also to verify that it has not been revoked under the time. This is done by downloading an up-to-date Certificate Revocation List (CRL) from the issuer of that certificate. These CRLs are updated at regular intervals. Devices can cache CRLs, but if the CRL validity time is short, the window of off-line usage will also be very limited.

In Chapter 5 the issues related to certificate usage and techniques for optimization of certificate revocation in ad-hoc environments are discussed. In Chapter 6 we present a CRL scheme that is optimized for ad-hoc networks.

We suggest two techniques for making off-line operation possible for a longer time. One is to allow applications to specify their own requirements for the freshness of the CRL. The other is to enable devices to share CRLs with each other.

A CRL distribution scheme optimized for wireless ad-hoc networks is presented. The scheme is based on a combination of over-issued delta CRLs and indirect CRLs, and also on two special techniques: a distribution point for all CRLs in a domain, and peer-to-peer exchange of CRLs between users in an ad-hoc network.
Chapter 5: Certificate Management in Ad Hoc Networks

5.1 Introduction

Ad-hoc proximity networks differ from wired local area networks in several important aspects [Roman 1997]. One aspect is that access to the global network can not always be established, either because of lack of coverage or because of high cost of communications. Furthermore, devices can remain unconnected for extended periods of time. Therefore, transactions and control mechanisms must be as independent from outside resources as possible. Another specific aspect is unpredictability of the network, that is, devices which form it are not preregistered or known in advance. Network is formed dynamically by users and devices in a proximity.

The possible unavailability of access to the global network has an important influence on the usage of public key certificates. Certificates rely on a Public Key Infrastructure [Arsenault 1999] consisting of a hierarchy of Certification Authorities (CA:s). Usage, management and validation of certificates usually assumes that a CA is available online. One of the most popular standards for public key certificates is based on in the X.509 standard [PKIX].

Previous work has been done on certification and authentication in ad hoc networks. One of the most difficult problems with using certificates for authentication in ad-hoc networks is related to the revocation of existing certificates. [Eschenauer 2002a] discusses the problem of certification in ad hoc networks versus the Internet, and offers some solutions to establish trust without using certificates. [Balfanz 2002] discusses different protocols for authentication in ad-hoc networks, also without the use of certification, but based on transmitting authentication information through a separate link (IR or contact between the nodes). [Hubaux 2001] describes a system for ad hoc networks similar to the PGP certification system ([Zimmerman 1995]) where certificates
are issued, stored and distributed by the users. [Burmester 1999] presents a theoretical model for using certificates and multiple paths of verification. An approach where the IP address of a node is calculated from its public key is presented in [Montenegro 2002]. [Venkatraman 2000] proposes an authentication scheme based on certificates, without mentioning the issues related to certificate revocation. [Candolin 2002] is also based on certificates, but here the nodes in the network may declare that other nodes have been compromised, thereby providing some form of certificate revocation, raising instead the issue of node misbehaviour and possibility of denial-of-service attacks based on maliciously accusing other nodes. Another solution based on delegating the revocation decision to the nodes is presented in [Crepeau 2003]. The authors here try to make malicious accusations more difficult, but the scheme still rests on trusting a network of nodes, and the revocation decision is still taken by the nodes, instead of a central authority.

Other solutions for certification in ad-hoc networks are based on the idea of threshold cryptography ([Shamir 1979, Chor 1985, Herzberg 1995, Shoup 2000]). [Zhou 1999] proposes using threshold cryptography for distributing the responsibility of a CA to several nodes in the ad-hoc network. Various solutions based on this idea are proposed and implemented in [Kong 2001], [Kong 2002], [Luo 2002] and [Yi 2003]. The weakness of schemes based on threshold cryptography is that on one hand they are computationally intensive, and on the other they rely on the cooperation of nodes in the network. Another problem is that for many applications, the enforcement of central policy decisions is necessary, thereby making a cooperative CA unsuitable.

The problems with most of the mentioned schemes is that they either do not solve the problem of certificate revocation, or they try to get around the unavailability of a CA by proposing different kinds of horizontal anarchical schemes, based on trust and reputation. While these solutions may be very useful in certain scenarios (routing protocols for example) they do not offer the adequate security solutions needed for most situations. Many of them focus on reputation, which is not useful for either identification or authentication except in very specific scenarios. There are many other scenarios (e.g. business, military) where using horizontal trust for sensitive transactions would be inadequate.

In this paper we explore techniques to relax the requirement of constant access to a CA, so that it is possible to use, manage and verify certificates even in situations where access to the Internet is not available. This has important advantages. The existing security mechanisms that are used (and well-proven) in standard networks can also be used in mobile ad-hoc networks. It becomes possible to enforce security policies, and the existing security policies defined for an organization can be applied to mobile ad-hoc networks, making these
environments, from a security point of view, an extension of the existing network.

5.2 Scenario

We will use the following scenario for exemplification.

Several devices form an ad hoc network in a place where access to the Internet is not available. Some of these devices may have been offline for a longer period of time. The assumption is that all of them have certificates which they have previously obtained from CA:s in the same certification hierarchy.

In the ad hoc network, the devices need to communicate and share information with various security requirements. They have to authenticate each other and to verify each other’s authorization to access the information. Transactions may also take place between the devices, like electronic payment transactions or signing digital contracts.

5.3 Online operation of certification systems

Prior to using them, certificates need to be obtained from a Certification Authority server. This is done by creating and submitting a certificate request to the CA. The CA verifies the request and issues the certificate, making it available to the requesting client. By means of a shared secret between the client and the server, the client can now fetch the certificate.

Once obtained, the certificates are used in communication with other entities.

When receiving the certificate of a peer, one needs to validate it to ensure its correctness. For this, access to an online CA is needed for several actions:

- obtaining the certificate chain up to a known CA
- verifying that the certificate has not been revoked
- getting updated policy information concerning the usage of the certificate

A certificate chain is needed in the case that the certificate to be verified comes from an unknown CA. In that case, the certificates of all the CA:s in the hierarchy up to a CA that is known and trusted are needed (Figure 5.1). The top CA of a certification hierarchy is known and trusted by all entities in that hierarchy, so usually a certificate chain contains all certificates up to the top CA.
A certificate may become invalid before its expiration date, for example if the corresponding private key has been revealed [Ford 1997]. In this case the certificate must be revoked, so it is no longer accepted as valid by anyone, even though it is signed by a trusted CA and is still in its validity period. This is done by adding the certificate to a Certificate Revocation List (CRL) [NIST 1995]. These lists are distributed through the entire certification hierarchy so that all Certification Authorities (CA) get them. When a client needs to validate a certificate, it contacts a CA to get the latest CRL and then checks if the certificate has been revoked or not [Adams 1999].

The policy of usage of the certificates may change over time as well. These changes must be communicated to all entities in that certification hierarchy.

### 5.4 Solutions for offline operation

Since the devices in an ad hoc network are not always offline, they can submit certificate requests and fetch the issued certificates from the CA servers while being online. The certificate functions that need to be available offline are the functions to validate a peer certificate.

To validate certificates offline, all information listed in section 3 must be available locally. Certificate chains are not an issue, since they can easily be stored by each entity together with their own certificates. When sending the certificate to another entity, the certificate chain is sent as well. Of course, since a certificate chain can be much larger in size than a single certificate, one can choose to send it only when it is needed, that is, when the peers do not have previous knowledge of the issuing CA.
A more difficult issue is the CRL and the policy information updates. These are updated at regular intervals, and are extremely important to the security of any application based on certificates. One does not want devices that have been offline for a while to accept certificates that have been revoked under the time, or to rely on outdated security policies. We call the CRL and the security policy changes the certification information updates.

When online, a certificate can be validated instantly, and the circulation of information regarding the revocation of a certificate or changes in security policy is fairly quick. Therefore, if a certificate is revoked in Sweden, it is probable that it will no longer be usable in Florida by the time its possessor gets there.

When offline, this situation changes. Even if a client has downloaded a CRL the last time it was connected to the Internet, it will expire quite soon, and there is no guarantee that the certificate it receives during a transaction has not been revoked in the meantime. Therefore, there is always a risk in validating certificates without online access. The risk becomes greater as the age of the available CRL increases, as more certificates will be revoked under that time.

For some applications this risk is not acceptable, but for others, the advantage in usability and convenience that is gained by allowing the CRL and security policy information to be somewhat outdated is worth the risk. We call the period that is acceptable, for an application, from the last update of the CRL and security policy, the grace period. Ad hoc aware applications that use certificates should therefore specify in their own security policies the grace period that is acceptable for a particular transaction. For example, monetary transactions up to a certain amount may be possible with a grace period of one week, while buying a car would require online access. Also, sharing certain sensitive documents would be possible with a grace period of a few hours, while for less sensitive documents the grace period may be one day.

This requires modifications to the certification module used by the devices, so that an application can request not only the validation of a certificate, but also specify the allowed grace period. The certification module stores the time of the last update and returns a negative response if the interval is longer than the grace period accepted by the application, indicating this reason.

This allows for a certain offline period during which applications can still use and verify certificates, according to their security policies.

5.5 Channelling update information

By adding a grace period to the policies of applications that use certificates, it becomes possible to use and verify certificates without direct access to a CA
server, if the last contact with a CA server was within the grace period of a particular application. However, if an entity has not had direct contact with a CA longer than the grace period, usage of certificates is impossible, even if other entities in the ad hoc network have the latest updates.

This is unnecessary limiting, since the information is there, in the ad hoc network, and we only need a secure way to spread it to the devices that don’t have it.

We call this technique *channeling of the update information*. It works in the following way. While online, a user obtains a signed CRL and security policy update from a CA, together with a timestamp (Figure 5.2). In the ad hoc network, the signed CRL and security policy update can be distributed to other devices, who can verify the signature of the CA to ensure their validity, and check the timestamp to see if they are within the required grace period.

![Figure 5.2: Obtaining the CRL from the CA](image)

The first part of the protocol looks like this:

- When in contact with a CA, the certification client asks for the current CRL.
- The CA server responds with the hash of the current CRL together with a timestamp, signed by the CA.
- The client checks the signature of the CRL (which can be a serial number or the hash of the CRL) and compares it to the one it already has. If the CRL differs, it requests the new CRL from the CA server.
- The server, if requested, sends the current CRL to the client. If there is any security policy update, it is also added to the package. This package is cryptographically signed by the CA server or by some other CA in the hierarchy.

By now each client has two items: a signed CRL (or a package containing both CRL and a security policy update), which can be verified by any entity in the certification hierarchy and a timestamp for that CRL, that shows the time when it was downloaded. Note that it is not necessary to download the entire CRL or policy update each time, only when it changes. This scheme can be optimized to further reduce network traffic by using delta-CRLs ([Cooper 2000]), CRL lists that only contain the changes from the base CRL.
This protocol is not loading the CA too much cryptographically, since a new timestamp only needs to be computed at certain intervals and than can be sent to all clients requesting updates.

![Figure 5.3: CRL exchange in ad hoc network](image)

When in an ad hoc network, peers can signal that their latest CRL is not within the grace period for a certain transaction (Figure 5.3). In that case the following protocol for exchanging the CRL can be performed between the device that does not have the required CRL (A) and the device that may have it (B):

- A sends to B the request stating the maximum age of the CRL, together with the signature of A’s current CRL.
- If B has a CRL that is new enough, it compares the signature of A’s CRL with the signature of its own. If the CRLs are the same, it sends to A the timestamp of the CRL, signed by the CA. If the CRLs are different, it also sends the CRL itself.
- A can now verify the signatures of the CRL and of the timestamp, and compare the signature of the CRL with the one contained in the timestamp. If the CA that issued the CRL is not known, it requests the corresponding certificate chain from B. Otherwise, the protocol is finished.
- B sends the CA certificate chain to A, if necessary.

The protocol can be adapted for group usage. An initial step is needed to find out which device has the newest CRL. After that, the exchange takes part in a similar manner with the two-party protocol.

One device may attempt to sabotage the process by pretending to have a CRL that is very recent, and then not sending it, thereby blocking other devices that have older, but still usable CRL:s. To prevent this, hashes of the CRL, together with a timestamp and signed by a CA, are used during the initial step (Figure 5.4). Also, if the CRL can not later be obtained from the device that was chosen during the initial step, the protocol is repeated using the second best result of the initial step.
- Each device informs the others about the CRL it has. This is done by sending a hash of the CRL with a timestamp, signed by a CA. After receiving the timestamps from all other devices, each device decides which one to request.
- Devices compare the hash of the newest CRL with their existing one. If they are the same, the process is finished, as they have a current timestamp for their CRL, signed by a CA. If necessary, they request the certificate chain for the CA that signed the timestamp. If the hashes differ, they request the new CRL, and also, if necessary, the certificate chain of the CA that signed it.
- The device that got chosen during the initial step distributes the CRL to the devices requesting it. If a device requires it, it also adds the certificate chain of the CA that signed the CRL.
- Recipients verify that the signature is correct. If verification does not succeed, they repeat the process with the second best result of the first step.

![Diagram](image)

Figure 5.4: Group protocol to exchange CRL

The group protocol is more efficient than repeating the two-party protocol for each device, since information regarding the CRL is already distributed during the first step. While in the two-party protocol it is the owner of the CRL that compares it with the client’s hash and decides whether to send it or not, here the clients obtain that information in the first step, and then only contact the owner in the case their CRL is different.
5.6 Conclusions

With the adaptations and modifications described above, usage and validation of public key certificates becomes more flexible and robust in ad hoc networks.

Considering the scenario presented in section 5.2, the devices can verify each other’s certificates by using stored CRL:s and certificate chains. The devices that have been offline for a longer period of time can request updated CRL:s and timestamps from the others. Information can then be shared, or transactions performed, if the maximum grace period allowed by each application has not been exceeded. Some of the constraints of normal certificate usage in ad hoc networks are in this way removed. Also, a balance between security and utility for each application can be specified as a part of the security policy.

The described scenario assumes that certificate chains of all users in an ad hoc network meet at some CA, which implies that all users belong to the same global PKI. The case when users belong to different PKI:s is discussed in the following chapter.
Chapter 6: 
Certificate Revocation System Based on Peer-to-Peer CRL Distribution

6.1 Introduction

Various types of certificates are basic tools of modern cryptography and network security. They are used in various protocols, in the form of public key identity certificates, binding a key to its owner or in the form of attribute certificates, being a proof of rights and capabilities of their owner.

Certificates have certain validity periods, decided by the policy of the issuing Certification Authority (CA). However, certificates can be revoked for various reasons before the expiration of their validity period. Such reasons may be that the private key related to the certificate has been compromised, or that the affiliation of the owner has changed and the certificate is therefore no longer valid, even though it has not yet expired [Ford 1997]. Validating certificates consists therefore not only of verifying the issuer's signature and the validity period, but also of checking the certificate revocation status [Adams 1999].

The usual approach for providing revocation information is based on Certificate Revocation Lists (CRLs). These lists are issued by the various Certification Authorities (CA), and contain the serial numbers and other information related to the revoked certificates that have been issued by respective CA [ITU 1997, McDaniel 2000]. Another technique is to verify the revocation status of a certificate by querying an on-line server able to provide this information [Myers 1999, Malpani 1999]. A good overview of different revocation schemes can be found in [Årnes 2000].

Various revocation systems have different characteristics when related to network traffic overhead, load on the servers that provide revocation information [Micali 1996, Willemsen 1999], freshness of the revocation information and suitability for off-line usage [Årnes 2000].
Recent trends in computer network environments, like mobile computing, wireless networks and ad-hoc networks [Roman 1997], make the suitability for off-line usage a more and more important quality of a revocation system. In [Morogan 2003b] different techniques to enhance off-line certificate validation are studied.

In this chapter we describe a distributed system for certificate revocation, based on peer-to-peer distribution of CRLs. The main goal of this approach is to achieve good off-line functionality and to reduce requirements on CRL distribution servers availability.

6.2 Certificate Revocation Lists

Certificate Revocation Lists (CRLs) are a widely used and accepted technique for distributing information about revoked certificates [McDaniel 2000]. However, they can be quite costly in terms of network resources and server load [Rivest 1998, Micali 1996].

The freshness or timeliness of the revocation information depends on the frequency with which the CRLs are updated. Timeliness will differ from system to system, depending on the particular policy of the certification infrastructure and on the sensitivity of the applications that use it. Standard CRL schemes can be very costly in situations where a high timeliness is required (e.g. sensitive transactions) or impractical in situations where clients may be performing these transactions off-line. Improved CRL schemes have been developed to solve some problems, or to optimize the functionality for a specific usage.

6.2.1 Basic CRL Scheme

The standard model for a CRL scheme [ITU 1997, Micali 1996] is shown in Figure 1. Here, a Certification Authority (CA) creates a CRL, and posts this CRL to a server, called CRL Distribution Point [Ford 1997, Adams 1999]. The end entities connect to the distribution point and retrieve the current CRL. This CRL can be cached by the end entities until it expires.

Each CA creates a CRL containing only revocation information about the certificates it has issued. Therefore, when an end entity receives a certificate from a peer, it will need to contact the distribution point for the peer certificate in order to obtain the CRL. Information about the address of the correct distribution point for a certificate can be stored in a field in the certificate [ITU 1997].

Each CRL is valid for a certain period of time, decided by the policy of the CA that has created it. This validity period can range from one month to one hour or even less, depending on the sensitivity of the transactions that the certificates are used for. The shorter the validity period, the higher the timeliness of the system.
Policies that require high timeliness will lead to high load on the distribution point and a high network traffic overhead, as many users in the system will frequently need to update their CRLs. Also, high timeliness means that cached CRLs on the clients will not be valid for a long time, so the off-line functionality of the system will suffer. Users will need to connect to the network often to update their CRLs.

![CRL distribution model diagram]

**Figure 6.1**: CRL distribution model.

There are some techniques that offer improvements over the basic CRL scheme. These improvements are mostly related to lowered load on the distribution points and increased timeliness.

### 6.2.2 Delta CRLs

Delta CRL is a way to reduce the size of the CRL updates. A delta CRL contains, as the name suggests, only the incremental changes that have occurred since the last complete CRL [Ford 1997]. Therefore, a delta CRL will be significantly smaller than a complete CRL (called here a base CRL), allowing a higher update frequency. In this scheme, the end user will cache the base CRL and only download the current delta CRL, leading to much smaller network traffic.

Delta CRLs are implemented by using the "Delta CRL Indicator" extension in the X.509 CRL Standard [ITU 1997]. The base CRL that it applies to is identified in the extension field.

Delta CRLs lower the network load significantly, making it possible to have a high update frequency of the revocation information.

### 6.2.3 Over-issued Delta CRLs

It is possible to issue multiple delta CRLs with overlapping validity times. In this way, the issuing CA does not need to wait for the expiration of an existing delta CRL, but can issue new ones with overlapping validity periods [Cooper 1999]. By doing so, peak load on the distribution point will be reduced, since not all delta CRLs cached by the users will expire at the same time.
Another advantage is that it is possible to make revocation information available sooner. Applications that need high timeliness can therefore download the latest issued delta CRLs, while others can continue to use older, but still valid delta CRLs. This makes it possible to have a more flexible revocation system, where applications that need timely information can get it, without forcing all other applications to do the same. By this differentiation a system with high timeliness can be created without putting an unrealistic load on the distribution points or increasing network traffic too much.

A characteristic of over-issued delta CRLs is that the CAs do not need to issue them at predefined intervals. Instead of issuing a new delta CRL every hour, for example, a CA could only issue a new delta CRL each time a certificate is revoked.

This leads however to a non-repudiation problem. A client may claim to have used an older (but still valid) delta CRL to accept a transaction, even though it had access to a newer delta CRL where the peer certificate was revoked.

### 6.2.4 Indirect CRL

In the basic CRL model, each CA issues a CRL with revocation information for the certificates it has issued. In an environment with many CAs, a client may need to download several CRLs from different distribution points to verify the certificates it receives.

Indirect CRLs are CRLs that are issued by a different authority than the issuer CA [Adams 1999], and they can be used to create a CRL containing revocation information from multiple CAs.

A client that needs multiple CRLs will be able to download all the information from one single distribution point, with less overhead than when downloading from different distribution points. Also, by caching it, clients will be able to verify certificates from all participating CAs without the need to connect to several distribution points.

This scheme has some disadvantages. One is that the resulting CRL can be quite large. Another disadvantage is that the expiration time of the resulting CRL will be equal to the shortest expiration time of the contained CRLs, which can lead to unnecessary network overhead. In the case of different expiration times and unsynchronized issuers, the resulting network overhead could be very large.

### 6.2.5 Freshest CRL

With over-issued CRLs and delta CRLs it is possible that even though a CRL is still valid, newer revocation information is available. A certificate extension
called "Freshest CRL" [ITU 1999] points to the freshest revocation information that can be obtained for that certificate. In this way, applications that have high timeliness demands can obtain that information, while applications with lower demands on timeliness can work without having to update their CRLs as long as they are valid.

6.3 Techniques for Off-line Verification of Revocation

All existing CRL schemes rely on contacting a CRL distribution point to download a valid CRL for a certificate that has to be verified. These CRLs can be cached by the client while being on-line, so certificates can be verified off-line as long as the cached CRLs are valid. If timeliness requirements are high, this period of off-line availability will be very short.

To increase the off-line functionality of CRLs, we propose the use of two techniques:

1. The use of CRL distribution points that can provide CRL information for all CAs in a domain. Clients can therefore cache CRL information not only from their issuer CA, but also from other CAs in the domain.

2. The use of a peer-to-peer CRL exchange protocol to extend the off-line functionality of CRL-based revocation schemes. Clients that have been off-line for a longer time will be able to download fresher CRLs and delta CRLs from clients that have had more recent contact with a distribution point. In this way, if one client in an ad-hoc network has valid CRLs, other clients will be able to download them, making authentication and other transactions possible.

6.3.1 Distribution Points for Multiple CRLs

We will first define the concept of revocation domain. We call a revocation domain a group of CAs whose clients are most likely to interact. For example, a revocation domain could be formed by all CAs in a city, or by all CAs in a company. The goal is to make it possible for clients to download and cache all revocation information available for the revocation domain they are currently in. In the case of a revocation domain based on geographical location, a localization service could be used by a client to find out which revocation domain it is in and the list of available CRL distribution points. In other cases, this information may be added to the security policy of the client.

CRL distribution points in the revocation domain will have access to all CRLs that are issued by CAs in the revocation domain. This can be implemented by having the CAs send their CRLs to all distribution points in the revocation domain they are part of. For increased scalability, a distribution point locator
service can be used by CAs to get the list of all distribution points in the domain. Distribution points announce themselves to the locator service. When a CA issues a new CRL, it contacts the locator service for a list of current distribution points, and then sends the CRL to all distribution points.

As a result, each distribution point in the domain will have all the latest CRLs from all CAs in the domain. A client can therefore retrieve all revocation information for the domain from any distribution point in the domain.

In existing CRL schemes, the client will connect a distribution point only when it needs the specific revocation information that the distribution point has. The information on what information is needed and where it can be retrieved lies on the client side. With distribution points having all CRLs in a domain, this is no longer the case, since the client does not have information about which CRLs are available. One solution would be to have the distribution points create one single Indirect CRL from all CRLs they receive. A client can then download this large CRL, which contains all revocation data for the domain.

![Figure 6.2 : Distribution Point for multiple CRLs and Locator Service.](image)

This however can be very inefficient. The CRLs in the domain can have different validity periods, and the CAs may issue updates with different frequencies. Each time such an update occurs, the distribution point would have to create a new CRL. A client requesting the latest updates would then have to download everything, even though it may already have most of the information.

A more efficient solution is to let the distribution point distribute the CRLs as they come from the CAs. Since the client does not have enough information to decide which updates it needs, the distribution point must be able to provide a list with the available CRLs to the client. This list will contain the issuing CA, the serial number, the issuing time and the expiration time of each CRL it has. The client will compare these numbers with the CRLs it already has and only request the missing ones or replacements for the ones that will expire shortly.
A problem for off-line functionality is related to the validity period of CRLs and the time they were issued. If a client downloads a CRL near the end of its validity period, the off-line usage window will be very short.

A solution to this problem is to have the CAs frequently issue overlapping delta CRLs. Having a high issuing frequency compared to the validity period of the delta CRLs insures that a client will always download CRLs that have a long validity time left.

To further improve off-line usage window, it is best to differentiate between applications with different levels of timeliness requirements. Making the delta CRL validity period longer will provide a longer window of usage for applications that do not require high timeliness, while providing frequent updates of the delta CRLs improves the timeliness for sensitive applications.

6.3.2 Peer-to-peer CRL exchange

Entities forming an ad-hoc network may have been off-line for different periods of time, so some of them will have fresher revocation information than others. With peer-to-peer CRL distribution, entities will be able to obtain fresh CRLs from another entity that has had recent contact with a distribution point. In this way, fresh CRLs will be propagated through the ad-hoc network, making it possible to verify certificates from all entities even after the cached CRLs of most entities have expired. This technique will be useful even in on-line operation, since it will significantly reduce load on the distribution points. In the on-line situation, an entity will only need to connect and download fresh CRLs from a distribution point if the entities it has performed transactions with did not have fresher CRLs.

The transfer of CRLs can take place immediately after the exchange of certificates. Figure 6.3 shows the steps of the CRL transfer scheme.

![Figure 6.3: Peer to peer CRL distribution protocol.](image-url)
In the first step, entities exchange a "Last CRL Update" value. This is the last time when each entity has updated their CRLs. By exchanging this value, it becomes possible for each entity to decide whether the other entity has fresher revocation information, without having to exchange the longer CRL lists.

In step two, entity A, after deciding that B has fresh revocation information, asks entity B for a list of all CRLs that entity B has. This list is similar to the one used for downloading from a distribution point (section 6.4.1). It contains, for each CRL, the issuer, the serial number, issuing time and expiration time.

In step three, entity A uses the list received from B to determine which CRLs to request, then it sends a request to B containing the issuer and serial number of each CRL it needs to download.

In step four, B sends the requested CRLs to A. A verifies their signatures to insure that they are valid, and use them to replace its expired or older CRLs.

Since all CRLs are signed by their respective issuing CA, their integrity and correctness can be verified by the end-entities, without regard to where they come from. One security issue characteristic to all CRL schemes that use over-issued CRLs is that a client can choose to use an older CRL even though it has access to a newer one. This is a non-repudiation problem when clients download the CRLs directly from their trusted distribution points. In the case of peer-to-peer CRL transfer, this is no longer only a non-repudiation problem, since clients can choose to only distribute certain CRLs and withhold newer ones. For example, if B's certificate has been revoked in the latest CRL update, B may still distribute CRLs from the previous update that are still valid. This problem is avoided for sensitive applications by requiring high timeliness for the CRLs. If the timeliness requirements are high, the older CRLs that B can provide will not be useful. Choosing the timeliness that an application requires for the CRLs is a matter of balancing the risks of accepting revoked certificates versus the increased off-line usability and the reduced load on the distribution points.

6.4 Conclusions

To illustrate the operation of this CRL scheme, we will use a scenario similar to the one presented in section 5.2. End-entities can be devices with permanent contact to the network and devices that connect to the network only occasionally. The domain contains a number of CAs and distribution points. The CAs hourly issue delta CRLs with validity times of one day. In this scenario, a device that updates its revocation information before going off-line will have valid revocation information for at least 23 hours. If this device is off-line for a longer period, it may be able to update its revocation information from other devices that have had contact with a distribution point during the last 23 hours. A device
that never goes on-line can still function and verify certificates by coming in contact with devices that go on-line frequently. A device with relatively fresh revocation information can therefore function as an "ad-hoc distribution point" for a network of devices from the same domain.

In conclusion, by having distribution points that can provide CRLs from all the CAs in a domain, together with peer-to-peer distribution of CRLs, a decentralized, flexible and distributed system for handling of revocation information is created. This benefits both on-line and off-line operation.

For on-line operation, the availability requirements on each distribution point will be reduced, since the same revocation information can be obtained from any distribution point in the domain. The load on the distribution points will also be reduced, since clients can obtain fresh CRLs automatically, through contact with other users, without having to download each update from the distribution point.

Off-line operation is also greatly improved. By downloading and caching all CRLs in the domain before going off-line, clients will have a longer period during which they can verify certificates off-line. Using peer-to-peer distribution, clients with more recent contact to a distribution point will refresh the revocation information of other clients.

Further steps in this research are to define the protocols discussed in the present paper in IETF RFCs. More exactly, the protocols to be defined are:

1. Protocol for downloading multiple CRLs from a distribution point
2. Protocol used for communication between the distribution point locator service, CAs and distribution points.
3. Peer-to-peer CRL transfer protocol.

Another interesting direction is to create simulations of different environments and analyze the characteristics of this system, especially the load on the distribution point, the peak access time to distribution points and the duration and network overhead of peer-to-peer CRL transfer.
Part 3:
Access Control for Documents in Ad-hoc Networks

With the GSO platform and the optimized certificate verification protocols, most of the security services available in classical networks become available in ad-hoc wireless networks as well. These services include authentication, confidentiality, integrity and non-repudiation. This part focuses on the issues related to the implementation of an access control system in wireless ad-hoc networks.

In order to function in an ad-hoc network environment, an access control system needs to be independent of centralized network resources such as a file server. We have designed an access-control system based on encapsulated documents and trusted decision engines [Morogan 2003d]. The idea here is to separate the access decision functionality from enforcement and storage. Documents move freely through the network, being protected by cryptographic encapsulation techniques. Their content is not accessible by users in this form. They contain, also inaccessible to the users, a set of use conditions or access control policies. These policies can be specified in one of the several existing access control policy languages.

To gain access to the contents of a document, a user needs to employ the services of what we call a “Trusted Decision Engine” (TDE). TDEs are small servers that are available on trusted servers throughout the network or on trusted hardware on the client machine. Their role is to decrypt the access control policy from a document and, based on the extracted policy, on the general system policy and on the users’ credentials, take an access decision. If the decision is to allow the requested access, the key needed to decrypt the requested data is given to the user.

The general architecture of the system is described in chapter 7. The structure of the documents is described in chapter 8, and the structure and functionality of a Trusted Decision Engine is described in chapter 9.
Chapter 7: Access Control for Documents in Ad-hoc Networks

7.1 Introduction

Ad-hoc proximity networks differ from wired local area networks in several important aspects [Roman 1997]. One aspect is that access to the global network can not always be established, either because of lack of coverage or because of high cost of communications. Furthermore, devices can remain unconnected for extended periods of time. Therefore, transactions and control mechanisms must be as independent from outside resources as possible. Another specific aspect is unpredictability of the network. That is, users and devices which form it are not preregistered or known in advance. Network is established dynamically by users and devices in a proximity. The mobility of the devices and their lower physical security influences the way sensitive data is treated as well – since sensitive data cannot be stored on such devices without protection.

Existing access control systems usually rely on a trusted authority in the form of a server to take the access decision and enforce it. Another characteristic is that usually, once access to a document has been granted, the document can be downloaded by the user and no further access control can be performed.

An access control system for ad-hoc networks should therefore have the following characteristics:

- It should be independent from network resources: since network access is not always available, access decisions should be made locally in a secure way. All the information necessary for making such decisions has to be available locally.
• Documents should be protected at all times: due to the low physical security of the devices, documents should not be stored in an unprotected form.

• Access control mechanism should be enforced at all times, even after a user has downloaded a document, enabling secure peer-to-peer transmission between users.

• Enforcement of complex access control policies: since the network is unpredictable, complex policies based on various attribute certificates of users can be necessary.

In this paper we propose a system which meets these requirements by expanding common document encapsulation techniques with access control capabilities.

7.1.1 Scenario

We will use the following scenario for exemplification.

Several devices form an ad hoc network at some location where access to the Internet is not available. For simplicity we assume that they are all from the same administration domain, so that the same authority is trusted to take access control decisions for all of them.

The users need to exchange sensitive documents with one another. Users may have access rights to some documents, or only to certain parts of them. We assume a topology without a central server for downloading, but only documents available locally that can be sent from peer to peer.

7.1.2 Existing approaches to access control in distributed systems

There are some interesting research results in the area of distributed authorization systems. [Johnston 1996] proposes a system based on use-conditions certificates for the protected resources and credential certificates for users. A policy engine compares the two and issues a ticket for the resource access controller. Another paper ([van Doorn 1996]) describes the design and implementation of secure network objects, and provides security for object-oriented network communication. The design accommodates both ACLs [Lampson 1974] and capabilities [Dennis 1966].

[Thompson 1999] describes the implementation of an access control system based on use-condition certificates for the resources and attribute certificates for the users. In this system, resources reside or are represented by servers. To access them, a client connects to the specific server and authenticates herself. The server then fetches the access policies, in the form of Use-Condition certificates, from
their prespecified locations, then checks user’s attribute certificates to see if the user fits the specified conditions.

The XML Access Control Language (XACL) provides a way to embed access control information in XML documents ([Hada 2000]), and [Damiani 2000] present an access control system that allows definition of access restrictions inside the XML documents.

All the above systems use a form of capabilities on the user side and conditions on the resource side, and as a result they are very flexible. The enforcement of an access control decision is based in all cases on a trusted server that can allow or deny access to a resource. In the case of documents, once access is permitted, the document is stored by the client in unprotected form. These characteristics make the systems reviewed above unsuitable for use in mobile ad-hoc networks. They fail to meet three of the requirements outlined in the previous section: they are not independent from network resources, relying on a trusted server for access control decision and enforcement. Documents are not protected at all times, making them susceptible to physical theft. Also, the access control system functions only when users access documents located on the server, not when users send documents to each other.

7.1.3 Implicit versus explicit access control

We can differentiate between two forms of access control, implicit and explicit access control [Ford 1994]. Encrypted data which can only be read by one specific user is a form of implicit access control. Users cannot access data unless they have the necessary key to decrypt it. In a system optimized for encrypting data for multiple users, this can function as a form of access control. The advantage of this approach is that enforcement of access control decisions is built-in. No separate mechanism is needed to decide whether the user has access to data or not, since she either has the key and can access it or she doesn’t have it. The disadvantages of this approach are that it offers very limited options with regard to the access control policy. There is no way to set more complex access conditions, like for example to require certain attribute certificates from the user. Also, it is not possible to handle other rights than the right to read data.

Explicit access control is based on an access decision that is taken by some trusted entity, based on an access policy for the system. The advantage of this approach is that it is possible to apply complex access control policies. Policies based on attribute certificates are possible, as well as handling of all read/write/create rights. The problem here is that the access control decision has to be enforced. This can be done in different ways. In a network, a server can be trusted to enforce the decisions of the access control system when responding to requests from users. If the data that has to be accessed is on the local computer of a user, it becomes more difficult to insure that the access policy is followed,
since in most cases, users can achieve total control over their machines and therefore an access control system running on them could be bypassed.

### 7.2 Techniques for access control for documents in ad-hoc networks

Ad-hoc network environments impose special requirements on self-sufficiency. Document encapsulation comes therefore as a handy solution. There are several document encapsulation standards, and they all provide document confidentiality, integrity and non-repudiation. We will look at ways to also add access control functionality. In this discussion, we will use PKCS#7 format [PKCS7] as a basis, since it is one of the most used encapsulation standards.

A general model of an encapsulated document is shown in Figure 7.1. The document is encrypted with a symmetric key and signed. The symmetric key is then encrypted with the public key of the recipient to form the RecipientInfo structure. Several RecipientInfos can exist, so the document can be encrypted for several users.

![Figure 7.1 : Document Structure](image)

In order to support multiple contents with different access requirements, we need to extend the model as shown in Figure 7.2. Each content is encrypted with a secret key and signed separately. Each recipient will have access to the appropriate keys, according to the access control policy of the document creator (Figure 7.3). Therefore this provides an implementation of implicit access control. RecipientInfo objects will be added for each user who has access to some section of the document, containing the appropriate keys for the content. A user will be able to verify the signatures for sections she has access to.

![Figure 7.2 : Content Info](image)
As stated in section 4, implicit access control provides very little flexibility for defining an access control policy. To make more complex access policies possible, encapsulated documents must support explicit access control as well.

![Recipient Info](image1)

**Figure 7.3**: Recipient Info

One common approach for distributed access control systems is to have a trusted server that takes the access control decision, and grant or deny access to the required resource [Thompson 1999, Neuman 1994]. For example, a user would authenticate herself to the server, present her credentials, and request access to a certain document. Based on users’s credentials and the access conditions of the document, the server would decide whether to grant access or not, and if yes, it would allow the user to access the document (Figure 7.4). In this situation, data resides on trusted servers, and users need to present their credentials in order to gain access to it.

![Decision Engine](image2)

**Figure 7.4**: Decision Engine

In ad-hoc networks, one cannot rely on external trusted servers to perform access control and to distribute documents, since such servers may not be availble. A user may find herself in an ad-hoc network with other users, wishing to share a document that the others need but cannot download from the trusted server. It is therefore important to have a possibility to implement a complex security policy, and make a system to inforce it even on untrusted machines. Documents should
then be protected from direct access, and users would need to show their credentials to the trusted decision engine in order to access them. The access they would gain would be limited to the parts that they have access rights to according to the access control policy. In other words, in ad-hoc network environments it is necessary to have a combination of implicit and explicit security – implicit in the way that access control policies cannot be bypassed, even when the document is available locally, and explicit in the way that complex policies can be enforced, requiring the user to present credentials that have to correspond to specific use conditions for different parts of the data.

Our approach is to implement a *Trusted Decision Engine*. This engine can be built into the operating system and administered by the superuser, or, much more securely, it can run on a separate trusted hardware device, like a smart card or an i-button [iButton, Loscocco 2001].

All trusted engines in a domain (one on every machine) contain the same asymmetric private key. Information encrypted with the corresponding public key can only be decrypted by the trusted engines. It is obvious then that placing the trusted engine in software on the user machine offers very little protection to the important private key, so a trusted hardware device is a much better solution. The trusted decision engine will accept requests from users, together with user’s credentials and the document’s use conditions.

![Trusted Engine Info](image)

**Figure 7.5** : Trusted Engine Info

The system functions in the following way: apart from the *UserInfo* objects containing keys for specific objects, there will also be one or several *TrustedEngineInfo* objects. The *TrustedEngineInfo* is encrypted with the public key of the respective trusted engine. It contains the necessary keys for the different parts of the document, together with lists of use conditions for each part (Figure 7.5). By adding multiple *TrustedEngineInfo* objects, access control information from multiple administration domains can be included.
When a user needs access to a certain section of a document, she will submit the request, together with the TrustedEngineInfo and the attribute certificates specifying her capabilities to the trusted decision engine (Figure 7.6). Based on that information, the decision engine will decide whether to give or not access to the user. If the answer is yes, then the decision engine will return the symmetric key that the user needs to access that document part.

In this way, documents can be accessed both by users who are specifically added as recipients, and by users who can prove their right to access the document via a Trusted Decision Engine.

7.3 Conclusions

We have described an access control system for documents in ad-hoc networks, based on document encapsulation. This system meets all the requirements set up in section 1:

- It is self-sufficient: documents contain all access control information that is necessary for a decision. Trusted Decision Engines can be made available on (most) devices in the network.
- Documents are protected at all times: documents are stored in their encapsulated form, and required data is extracted from them only when requested.
- Access control mechanism is enforced at all times: when another user needs access to the document, it can simply be sent to her. If the peer doesn’t have the necessary keys, or the required attribute certificates to “convince” the trusted decision engine to decrypt the data, she will not be able to access the document.
• Enforcement of complex access control policies is possible: using access conditions for data, complex requirements can be specified. Users can show their rights to access the data by supplying the required attribute certificates to the trusted decision engine, and this represents a flexible capability system.

Considering the scenario presented in section 7.1.1, users can exchange documents and use their locally stored capabilities for accessing them. The decision engine can then extract the data stored in documents and user’s attribute certificates and take an access decision based on it.

Some of the problems with sharing sensitive documents in ad-hoc networks are in this way removed. Further research includes proposing extensions to the PKCS#7 and XML format in order to support access control information and a feasibility study on the deployment of Trusted Decision Engine smart cards.
Chapter 8: Access Control System Based on Encapsulated Documents

8.1 Introduction

Mobile computing and wireless ad-hoc networks are trends that pose new stronger requirements on the flexibility and distribution of computing systems ([Roman 1997]). In particular, security mechanisms that are optimized for wired local area networks need to be adapted to the new environments.

There are several important aspects where the wired local area network environment differs from the wireless mobile networking one. One aspect is that in wireless networks, access to the global network can not always be established, either because of lack of coverage or because of high cost of communications. Furthermore, devices can be disconnected for extended periods of time. Therefore, transactions and control mechanisms must be as independent from outside resources as possible. Another specific aspect is unpredictability of the network. That is, users and devices which form it are not preregistered or known in advance. Network is established dynamically by users and devices in a proximity. The mobility of the devices and their lower physical security influences the way sensitive data is treated as well – since sensitive data cannot be stored on such devices without protection.

An access control system designed for mobile ad-hoc networks has to take all these characteristics into account. Our purpose is to design an access control system that is entirely distributed, and where reliance on resources in the network is reduced to a minimum. This system, when used in a wired network, would have the advantage of reduced load on central servers, while in a mobile ad-hoc network it would continue to be in effect even without outside connectivity.

In the following sections, we will outline the requirements that need to be met by an access control system designed for operation in wireless ad-hoc networks,
then we will present a system that meets these requirements. The focus of the chapter is proposing some extensions to the XML standard ([Bray 2000]) and the PKCS#7 standard ([PKCS7]) that will enable XML and PKCS#7 documents to contain access control policy information and enforce its implementation.

**8.1.1 Requirements for a distributed access control system**

Analyzing the special conditions in mobile ad hoc networks, we have formulated a list of requirements that an access control system should meet:

- It should be independent from network resources: since network access is not always available, access decisions should be made locally in a secure way. All the information necessary for making such decisions has to be available locally.

- Documents should be protected at all times: due to the low physical security of the devices, documents should not be stored in an unprotected form.

- Access control mechanism should be enforced at all times, even after a user has downloaded a document, enabling secure peer-to-peer transmission between users.

- Enforcement of complex access control policies: since the network is unpredictable, complex policies based on various attribute certificates of users can be necessary.

**8.1.2 Model of the access control system**

In the previous chapter we have described techniques and the general architecture of an access control system for ad-hoc and distributed networks. Instead of relying on central trusted servers to make and enforce access control decisions, the system we propose uses small policy decision engines that can run on trusted hardware on the client machines (e.g. smart cards), or as services on trusted servers throughout the network. We call these Trusted Decision Engines (TDEs). They are not used for storing and protecting the documents, instead the documents are protected at all times by cryptographic encapsulation techniques. All TDEs in a domain share one pair of asymmetric cryptographic keys, which they use to access, sign and verify encapsulated documents. We call this pair of keys the TDE asymmetric keys for the domain. The private key of the pair is available to all the TDEs in the respective domain and only to them.

When a document is created, in order to make it part of the access control system, it is sent, together with its particular use conditions, to a local TDE. The TDE verifies the rights of the user to create the document, according to the
general domain policy and the users’ credentials. If the action is permitted, the TDE creates an encapsulated document, locked and signed with the TDE keypair. More exactly, the TDE will encrypt the data of the document with a randomly generated symmetric key, then encrypt the symmetric key and the related security policy with the TDE public key for that domain, as well as sign the document with the TDE private key. Therefore, only other TDEs in a domain are able to access the document, and it can be verified to be signed by a TDE. The resulting encapsulated documents can then be distributed freely through the network.

![Figure 8.1: Trusted Decision Engine](image)

To access a document, the user will send this document to a locally available TDE. The TDE will decrypt it using the TDE private key and extract the access control policy. According to it, it may request further credentials or authentication from the user. Based on the document policy, on the TDE’s policy and on the information presented by the user, the TDE will take the decision whether to grant or not access to the document (Figure 8.1). If access is to be granted, the TDE will send the user the symmetric key used to encrypt the document data. In this way, the user will never have access to the access control policy itself, but only to the data, or, for multi-part documents, the parts of data of the document that it is entitled to.

The advantages of this system are that it separates the decision functionality from other parts of the access control system: policy storage and enforcement. The TDEs are not used to store policy information for each specific document, which instead is stored in the document itself. Also, TDEs are not used to store and protect the documents, which instead are protected at all times by cryptographic encapsulation. The part of the access control system that needs to operate on a trusted platform is therefore minimized, making it possible to implement the TDEs as a generic service, available in multiple ways – as a number of servers on the wired network, or running on trusted hardware like a smartcard on the client machine.
For this system to work, we need a way to add access policy information to encapsulated documents. The access policy information has to be stored in such a way that it is not possible by users to access the policy information, bypass it or tamper with it in any way. The focus of this paper is to propose and analyze extensions for securely adding access control policy information to two of the mostly used document encapsulation standards: the XML document standard and the PKCS#7 encapsulation standard.

8.2 Access Control for XML Documents

There are several standardized elements that can be used to add security functionality to an XML document. For our purposes, we are going to look at three standards, XML Encryption, XML Signature and XML Access Control Language.

XML Encryption ([Imamura 2002]) provides the means to encrypt data in an XML document with various levels of granularity. It is possible to encrypt a whole document, particular elements in a document, or the content of elements. The key element for encryption is EncryptedData, which is used to hold the ciphertext and related information. The key used for encryption can be stored in an EncryptedKey element.

XML Signature ([Solo 2002]) specifies the rules and syntax for digitally signing XML documents. A signature can be calculated for specific elements or for the entire document. The Signature element is used to store the digest for the signed data, the signature value and other related data. The standard also defines elements for different asymmetric keys.

XML Access Control Language ([Hada 2000]) is used to specify an access control policy for an XML document. The language is based on a provisional authorization model ([Jajodia 2000]). The granularity of the policy can be as fine as single elements within the document.

The existing access control model presumes that the XACL policy is either stored in the document or fetched from some other location by the XACL processor, while the document itself is stored in a protected place. When a user requires to perform an action on a document, the XACL processor will take an access decision and allow or deny that action. In our scenario, documents are not stored in a place controlled by the XACL processor, but are available to the users directly. To protect them, they are encrypted and signed. While the documents can be enveloped for specific users, a general access control system is much more powerful. We will now define a system for adding XACL access control information, together with the keys necessary to access the document, in such a way...
way that any TDE in the domain can be used to perform an access control decision.

8.2.1 Encapsulated Access Control for XML Documents

Our goal is to define a way of adding access control policies to XML documents in such a way that they are only accessible to the correct TDEs and are protected from tampering.

An XML document may contain a number of EncryptedData elements for which we want to provide access control through a TDE. Each of these EncryptedData elements may be encrypted with a different symmetric key and may have a specific AC policy. There may also be an AC policy for the whole document.

We define two new elements, UnitPolicy and DocumentPolicy. A UnitPolicy element contains the keys for an encrypted element within the document, together with the element policy and a reference to the element itself. Instead of a single reference, we can have a list of references, to support several elements encrypted with the same key and having the same policy. All UnitPolicy elements for a document, together with the policy for the whole document, form the DocumentPolicy element.

```xml
<DocumentPolicy Id="TDEPolicy">
  <UnitPolicy Id="part1">
    <ReferenceList>
      <DataReference>#part1</DataReference>
    </ReferenceList>
    <Policy><xacl>....</xacl></Policy>
    <KeyValue>xyzabc</KeyValue>
  </UnitPolicy>
  <UnitPolicy Id="part2">
    ....
  </UnitPolicy>
  <Policy><xacl>....</xacl></Policy>
</DocumentPolicy>
```

The DocumentPolicy element has now to be encrypted so that it is only accessible to the TDEs in a domain. A symmetric key is randomly generated and used to encrypt the contents of the DocumentPolicy element into an EncryptedData element. The EncryptedData element contains a reference pointing to the location of the encrypted symmetric key, which is encrypted with the public key of the TDE and added to the document. Finally, a signature is calculated for the DocumentPolicy element and added to the document. The structure of an XML document containing encapsulated access control information is shown in Figure 8.2.
8.2.2 Access control process

When a user needs to access a restricted field in an XML document, she can check if there is an EncryptedKey element which can be opened with her private key. If not, the services of a TDE have to be used.

The TDE will check for a DocumentPolicy and EncryptedKey that it can access. If such a DocumentPolicy exists, it will decrypt the policy encryption key and will use it to decrypt the DocumentPolicy element. The TDE will thus have access to the access control policy and to the content-encryption key for elements in the document. The next step will be to verify the digest and the signatures included in the Signature element. If verifications are correct, the TDE can proceed to implement the access control policy for the requested element, granting or not access to the user based on user credentials or other requirements (Figure 8.3).

Once the access decision is taken, granting read access to the message can be done in two ways. One is to simply send the content-encryption key for the element to the user, by some secure channel. The user will then be able to open its copy of the document. Another method is to create an EncryptedKey value for the specific user with the key to the requested element, and append it to the document. In this way, the user will be able to access the document at any time, without the need to access the TDE or to store a sensitive content key.
8.2.3 Security of the system

We will analyze how secure this system is. We have identified three specific threats:

1. Disclosure of access control information, which may be sensitive and not available to users.

2. Modification of access control information – even if the information is not accessible to users, an attacker has control over the protected message, and may try to modify the portion of data that contains the access control policy, so that the TDE will take the wrong decisions and give unauthorized access.

3. Replacement of access control information – an attacker might try to replace the part containing the access control policy with another access control policy, keeping the content-encryption key in place.

Disclosure of access control information is prevented by encrypting the `DocumentPolicy` value with a policy-encryption key and then encrypting the policy-encryption key with the public key of the targeted TDE. As long as the private key of the TDE is secure, an attacker will not be able to get access to the policy-encryption key. Note that the policy-encryption key is used only for encrypting the `DocumentPolicy` of a particular domain, and is different from the content-encryption key. Therefore, other users who may have direct access to the message (by having access to other `EncryptedKey` elements) will have access to the content-encryption key but not to the policy-encryption key.

The `DocumentPolicy` is protected against modification by being signed by the originator TDE. If such a signature is missing, the TDE will not allow any access, so any modification would not produce results. Since the `DocumentPolicy` contains both the access policy and the keys necessary for access, it is not possible to modify only one part of it without rendering the signature invalid.

8.3 Access Control for PKCS#7 Documents

There are two PKCS#7 content types that envelope data, `EnvelopedData` type and `SignedAndEnvelopedData` type ([PKCS7]).

The structure of the `EnvelopedData` type is shown in Figure 8.4. The input is encrypted with a random generated content-encryption key and together with a content type value and an algorithm identifier form the `EncryptedContentInfo` value. For each recipient, the content-encryption key is encrypted in its turn with the public key of the recipient. A `RecipientInfo`
value is created for each recipient. The RecipientInfo contains the unique identification of the recipient’s certificate, an identification of the algorithm used to encrypt the key, and the encrypted key.

![RecipientInfo Diagram]

**Figure 8.4 : Enveloped Data**

The structure of the SignedAndEnvelopedData type is similar to the EnvelopedData, but also contains signatures for the content (Figure 8.5). The process for encryption is similar to EnvelopedData.

![SignedAndEnvelopedData Diagram]

**Figure 8.5 : SignedAndEnvelopedData**

8.3.1 Extending the PKCS#7 standard

The PKCS#7 standard supports a basic way of including access information for multiple recipients by including multiple RecipientInfo values.

In order to add full access control capabilities, we will define an extended RecipientInfo type and an AccessPolicyData content type.

The AccessPolicyData content type is defined as follows:

```plaintext
AccessPolicyData ::= SEQUENCE {
  version Version,
  documentKey DocumentKey,
  acPolicyFile ACPolicyFile,
  digest DigestInfo }

DocumentKey ::= OCTET STRING

ACPolicyFile ::= OCTET STRING
```

AccessPolicyData content type consists of the clear-text content-encryption key of the document, together with the access control policy file, and a PKCS#7 digest for both.
The extended \texttt{RecipientInfo} type, apart from the encrypted key and recipient-related info, will also contain an optional \texttt{AccessPolicyInfo} field. If the \texttt{RecipientInfo} is targeted to an end user, the \texttt{AccessPolicyInfo} field will not be added and the \texttt{EncryptedKey} field will contain the content-encryption key necessary to decrypt the contents of the message, as is the case in standard PKCS#7 messages.

For access control purposes, the \texttt{EncryptedKey} field of the \texttt{RecipientInfo} will not contain the content-encryption key of the message, but a different symmetric key, which we will call the policy-encryption key. The policy-encryption key can be used to decrypt the \texttt{AccessPolicyInfo} field. The \texttt{AccessPolicyInfo} field contains the \texttt{AccessPolicyData} value for the document, encrypted with the policy-encryption key and signed by the policy issuers.

The structure of the extended \texttt{RecipientInfo} type is shown in Figure 8.6, and its ASN.1 definition is as follows:

```
RecipientInfo ::= SEQUENCE {
  version Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  keyEncryptionAlgorithm
  KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey,
  accessPolicyInfo
  [0] IMPLICIT SignedData OPTIONAL}
```

This is the definition of \texttt{RecipientInfo} from [PKCS7], with the addition of the optional \texttt{AccessPolicyInfo} field. All the data types are as defined in [PKCS7].

![Figure 8.6: Extended RecipientInfo](image)

The process of creating a \texttt{RecipientInfo} value for access control is the following:

- A policy-encryption key is generated at random.
- The policy-encryption key is encrypted with the TDE’s public key.
• A DigestInfo value is calculated for the document’s content-encryption key, together with the access control policy file of the document.
• The document’s content-encryption key, together with the access control policy file of the document and the DigestInfo, are collected into a AccessPolicyData value.
• The AccessPolicyDate value is encrypted with the policy-encryption key into a PKCS#7 EncryptedData value.
• The AccessPolicyInfo value is obtained by creating a PKCS#7 SignedData value from the EncryptedData obtained in the previous step. The signers are, according to the policy of the system, the creators of the message or the administrators of the system.
• The encrypted policy-encryption key, together with TDE specific information and the AccessPolicyInfo are collected into an extended RecipientInfo value.

By only modifying the RecipientInfo structure, it becomes possible to add access control information to a message using existing data types, like EnvelopedData and SignedAndEnvelopedData. By adding multiple RecipientInfo values to a message, it is possible to combine generic access control based access to data with enveloping data for specific users. It is also possible to add multiple RecipientInfo values for multiple access control domains, specifying the correct access control policy in each domain.

8.3.2 Access control process

When a user receives an encapsulated object, she can check if there is a RecipientInfo value which corresponds to her private key. If not, the object has to be sent to a TDE in order to be accessed.

![Figure 8.7: Access control process](image)

The TDE will check for a RecipientInfo corresponding to its private key. If such a RecipientInfo exists, it will decrypt the policy-encryption key and will
use it to decrypt the `AccessPolicyData` value. The TDE will thus have access to the message policy and to the content-encryption key. The next step will be to verify the digest included in the `AccessPolicyData`, and then the signatures of the `AccessPolicyData`. If verifications are correct, the TDE can proceed to implement the access control policy of the message, granting or not access to the user based on user credentials or other requirements (Figure 8.7).

Once the access decision is taken, granting access to the message can be done in two ways. One is to simply send the content-encryption key to the user, by some secure channel. The user will then be able to open its copy of the document. Another method is to create a `RecipientInfo` value for the specific user to the document. In this way, the user will be added to the list of recipients, and will be able to access the document at any time, without need to access the server or to store a sensitive document key.

### 8.3.3 Security of the system

We will analyze how our suggested extension to the PKCS#7 standard works from the point of view of security. We have identified three specific threats:

1. Disclosure of access control information, which may be sensitive and not available to users.
2. Modification of access control information – even if the information is not accessible to users, an attacker has control over the protected message, and may try to modify the portion of data that contains the access control policy, so that the TDE will take the wrong decisions and give unauthorized access.
3. Bypassing of access control information – an attacker might try to replace the part containing the access control policy with another access control policy, keeping the content-encryption key in place.

Disclosure of access control information is prevented by encrypting the `AccessPolicyData` value with a policy-encryption key and then encrypting the policy-encryption key with the public key of the targeted TDE. As long as the private key of the TDE is secure, an attacker will not be able to get access to the policy-encryption key. Note that the policy-encryption key is used only for encrypting the `AccessPolicyData` of a particular domain, and is different from the content-encryption key. Therefore, other users who may have direct access to the message (by having their specific `RecipientInfo` values) will have access to the content-encryption key but not to the policy-encryption key.

The `AccessPolicyData` is encrypted with a symmetric key and signed. If the system policy mandates a signature by a specified authority, modification of the access control policy is impossible. However, if this is not the case, an attacker might attempt to modify the access control policy by trial and error, and then
compute a new signature. This possibility is prevented by the inclusion in the `AccessPolicyData` structure of a digest of the access control policy and content key. Modifications in the encrypted `AccessPolicyData` will be discovered by the TDE when verifying the digest.

The digest also protects against replacement of the access control policy with another, since it is computed for both the access control policy and the content-encryption key, and then encrypted with the policy-encryption key.

### 8.4 Conclusions

By defining some additions to the XML and to the PKCS#7 standards, it becomes possible to include protected access control information in encapsulated documents. This information, together with the key for the content, is available to the TDEs in a specific domain, and is protected from tampering or disclosure to the other users. It is possible to add access control information for multiple domains, as well as to add key information for specific users, resulting in a highly flexible system. Implementing access control in this way, the requirements that have been set up in section 8.1.1 are met:

- It is independent from network resources: documents contain all access control information that is necessary for a decision.

- Documents are protected at all times: documents are stored in their encapsulated form, and required data is extracted from them only when requested.

- Access control is enforced at all times: when another user needs access to the document, it can simply be sent to her. If the recipient doesn’t have the necessary keys, or the credentials required by the trusted decision engine to decrypt the data, she will not be able to access the document.

- Enforcement of complex access control policies is possible: using access conditions for data, complex requirements can be specified.

This system protects documents against access from users without the required credentials. It should be made clear though that it does not prevent users from saving a document to which they have access in cleartext and then distributing that copy. No system can protect against that as long as the users machines are not trusted platforms. In conclusion this system can be used for access control but not for digital rights management types of usages.

There are several interesting directions for further research from here. One is to create and submit proposals for standardizing the required additions to XML.
The other is to implement a test system with documents and TDEs and perform usability/performance studies, especially with the TDE on a trusted hardware device like a smartcard.
Chapter 9: Enforcing Access Control Policies in Ad-hoc Networks

9.1 Introduction

Our approach for an access control system is to have the documents protected at all times during transfer and storage against unauthorized access, and delegate the access control decisions to a number of trusted decision engines available on trusted servers or on trusted hardware devices in the users computers. When a user receives a document, she will need to use the services of such a trusted decision engine to gain access to it.

In chapter 7 we have described a general architecture for an access control system that can function in an ad-hoc wireless network environment. In this chapter we will analyze one important element of this system, the Trusted Decision Engine.

9.1.1 Model of the access control system

The access control system is based on authorization servers that run on trusted hardware devices [iButton, Loscocco 2001] which we call Trusted Decision Engines (TDE). The TDEs perform and enforce the access control decision on behalf of the user, according to the policy of the documents and of the system.

Ad-hoc network environments impose special requirements on self-sufficiency. Document encapsulation comes therefore as a handy solution. There are several document encapsulation standards, and they all provide document confidentiality, integrity and non-repudiation. We will look at ways to also add access control functionality. In this discussion, we will use PKCS#7 format [PKCS7] as a basis, since it is one of the most used encapsulation standards.
A general model of an encapsulated document is shown in Figure 9.1. The document is encrypted with a symmetric key and signed. The symmetric key is then encrypted with the public key of the recipient to form the RecipientInfo structure. Several RecipientInfos can exist, so the document can be encrypted for several users.

To make more complex access policies possible, we add support for explicit access control to the encapsulated documents.

To achieve that, apart from the RecipientInfo objects containing keys for specific objects, there will also be one or several TrustedEngineInfo objects. The TrustedEngineInfo is encrypted with the public key of the respective trusted engine. It contains the necessary keys for the different parts of the document, together with lists of use conditions for each part (Figure 9.2). All TDEs in a domain have access to a common private key. By adding multiple TrustedEngineInfo objects, access control information from multiple administration domains can be included.

To access a document, the user will send this document to the TDE. The TDE will decrypt it using its private key and extract the access control policy. According to it, it may request further credentials or authentication from the user. Based on the document policy, on the TDE’s policy and on the information
presented by the user, the TDE will take the decision whether to grant or not access to the document (Figure 9.3). If access is to be granted, the TDE will send the user the key necessary to decrypt the document.

![Trusted Decision Engine](image)

**Figure 9.3**: Trusted Decision Engine

### 9.2 The Trusted Decision Engine

We will now describe the functioning and structure of a TDE. Since the TDE contains the private key on which the whole enforcement of access control policies in a system depend, it has to reside on some form of trusted device, like a smartcard or an i-button [iButton, SCBasics]. The implementation will therefore consist of a client, residing on the host computer, and the TDE itself, running on the trusted device.

When the client is instantiated, or when the TDE first becomes available to the client, a handshake is performed during which the client obtains relevant information such as the domain or domains that the TDE belongs to. This information will be used by the client when selecting the relevant TDEUserInfo.

The client provides a method, SubmitTDEQuery, which other programs can call for submitting an access query to the TDE. In Java, the syntax for SubmitTDEQuery is the following:

```java
SymmetricKey SubmitTDEQuery(
    EncapsulatedDocument doc,
    String action,
    Credentials credentials)
throws InsufficientCredentialsException
```
The parameters are the encapsulated document for which access is sought, the action required, and a collection of the user’s credentials relevant for getting access to the document. In case the TDE grants the access right, `SubmitTDEQuery` returns a `SymmetricKey` object that can be used to decrypt the payload of the encapsulated document.

Once the `SubmitTDEQuery` is called, the client extracts the relevant `TDEUserInfo` for the domain from the encapsulated document, and sends it to the TDE, together with the requested `Action` and the collection of credentials (Figure 9.4).

The TDE now has access to all information it needs to perform an access decision. It uses the `TDEPrivateKey` for the domain to decrypt the `TDEUserInfo` and extracts the access control policy applicable to the particular document, together with the `SymmetricKey` that is needed to decrypt the payload. By comparing the requirements of the policy with the existing credentials, it can take a decision whether to grant or not access. If access is to be granted, the `SymmetricKey` is returned to the client, or else just a null value.

![TDE Structure](image)

**Figure 9.4 :** TDE Structure

If the client receives a null value from the TDE, it triggers a `InsufficientCredentialsException`. Otherwise, the `SubmitTDEQuery` method returns the `SymmetricKey` that the calling program can use to decrypt the payload of the document.

In the case of a multi-part document (not supported by PKCS#7 but supported by for example XML) the TDE may decide that the user has the right to perform the requested action on several parts of the document, resulting in multiple `SymmetricKey` objects. In this case, the TDE will first communicate to the client the number of keys to be returned, then proceed to sending them. Returning the value 0 indicates that the user does not have access rights to any part of the document.
The syntax of SubmitTDEQuery for multi-part documents is then:

```java
SymmetricKeys SubmitTDEQuery(
    EncapsulatedDocument doc,
    String action,
    Credentials credentials)
throws InsufficientCredentialsException
```

In this case, the object returned from the method is SymmetricKeys, an enumeration of SymmetricKey objects.

### 9.3 Conclusions

By using the extended RecipientInfo type, it is possible to include access control information to encapsulated documents. This information, together with the key for the content, is sent by the Trusted Engine Client to the TDEs in a specific domain. Several RecipientInfos can be included, making it possible to specify different access control policies for different domains, and also to make the document available directly to specific users, resulting in a highly flexible system. By encrypting and signing the access control information for the specific TDE, it is protected from tampering or disclosure to other users or TDEs from other domains.

Programs can easily access the TDE by calling the SubmitTDEQuery on the Trusted Engine Client, and obtaining the required symmetric key. The client also contributes to the efficiency of the system, by only sending the TDEUserInfo instead of the whole document. In this way, the quantity of data sent to the TDE is the same, regardless the size of the payload.

Implementing access control in this way, the requirements that have been set up in section 1 are met:

- It is self-sufficient: documents contain all access control information that is necessary for a decision.
- Documents are protected at all times: documents are stored in their encapsulated form, and required data is extracted from them only when requested. When another user needs access to the document, it can simply be sent to her. If the peer doesn’t have the necessary keys, or the credentials required by the trusted decision engine to decrypt the data, she will not be able to access the document.
- Enforcement of complex access control policies is possible: using access conditions for data, complex requirements can be specified.
Chapter 10:
Conclusions and Future Directions of Research

10.1 Conclusions

Our work has concentrated on investigating problems related to the implementation of a security system in the environment of wireless ad-hoc networks.

The first part of the thesis is the design and implementation of a security platform based on generic secure objects. The flexible and modular nature of this platform makes it suitable for deployment on devices that form ad-hoc networks – ranging from Java-enabled phones to laptops and PDAs.

The second and third parts investigate the problems that appear when implementing in ad-hoc networks some of the security technologies that are standard building blocks of secure systems in classical computer networks. Two such technologies have been found to present problems, namely the areas of certification and of access control. In a series of articles, we have described the problems that appear and devised solutions to them by designing protocols, techniques and extensions to standards that are optimized for usage in the ad-hoc network environment.

Using the techniques developed, it is possible to provide the standard security services, as devised by OSI, in ad-hoc networks. Secure usage of certificates, together with the underlying platform based on secure objects, makes possible the implementation in a range of devices of privacy, authentication and non-repudiation services. Access control becomes possible through the usage of the trusted decision engine system, together with identity and attribute certificates. Also, the availability of the whole system is increased due to lower reliance on centralized network resources.
10.2 Further Research

There are some interesting research directions that can be pursued from here, relating to both the Generic Security Object platform described in Part 1 of the thesis and to the various protocols described in Part 2 and 3.

10.2.1 Moving to a Component Based Design

The Security Objects have been designed to be very flexible and modular, a property which makes it easy to create custom systems for different platforms and security requirements. However, being implemented as Java objects, once the applications have been compiled, the system is locked. The objects cannot easily be changed or replaced without changing the whole program. It would be desirable to have even more flexibility, allowing to dynamically download and instantiate an object when needed, make an application distributed when resources are low, or only load the parts that are needed for a particular operation (Figure 10.1). With the current implementation, all this is only possible by creating a new application, tuned specifically for those new requirements. Therefore it would be interesting to go one step further in the direction of modularity and flexibility by using the Secure Objects to create independent Security Software Components.

Components are encapsulated objects that comply with a standardized execution and communication architecture. They can execute independently, and they communicate through a standardized interface. A component is developed for a specific task but not for a specific application. Therefore, once a component is defined, it becomes an independent piece of software that can be used in many different applications. The opposite is also true, that an application that uses a certain component could also use another one that provides the same functionality.

![Diagram](image)

**Figure 10.1**: Dynamic use of security components.
To be useful and easy to reuse, the components have to be created according to some standardized architecture. Some of the architectures existent today are JavaBeans [Calder 1999, Colan 1999], Enterprise JavaBeans [Roth 2000], CORBA [CORBA] and Microsoft DCOM [Brown 1998, Horstmann 1997].

10.2.2 Standardising the protocols

The creation of standards for the various protocols described in the thesis would be another direction to pursue, enabling their usage and testing on a large scale. The protocols to be standardised are:

- peer-to-peer CRL transfer protocol described in chapter eight
- encapsulation of access control information in XML documents described in chapter ten
- TDE structure and API described in chapter eleven

10.3.3 Deployment of the complete system

Another direction is the implementation and deployment of these protocols and techniques, based on the Generic Secure Objects platform and using a range of hardware devices and network configurations. Such deployments can be used to perform tests regarding the usability and performance of the protocols, measuring the load on the network, off-line usage window in the case of CRL verification, and the load on servers when devices are on-line. Other interesting tests would be related to the performance of hardware device based TDE servers, determining the limitations and usability of the TDE based access control system.
Appendix:
Generic Security Objects

A.1 Identification and Authentication Objects

A.1.1 DistinguishedName

A DistinguishedName is an identification for a user that is unique in a specific context [X500]. To the DistinguishedName there usually is a corresponding asymmetric key pair, and the two are linked together by a certificate. The DistinguishedName object contains the following fields:

- countryName
- stateOrProvinceName
- localityName
- organizationName
- organizationalUnitName
- commonName
- emailAddress
- urlAddress

A user may have several DistinguishedName:s, with different attributes (for example one from work and another one from his bank). All DistinguishedName:s of a user are stored in the registration file. A new DistinguishedName can be created with one of the constructors:

DistinguishedName() or
DistinguishedName(String countryName,
    String stateOrProvinceName,
    String localityName,
    String organizationName,
    String organizationalUnitName,
    String commonName,
    String emailAddress,
    String urlAddress)

All the fields in the DistinguishedName object can be accessed with the methods:
String get_countryName()
String get_stateOrProvinceName()
String get_localityName()
String get_organizationName()
String get_organizationalUnitName()
String get_commonName()
String get_emailAddress()
String get_urlAddress()

A DistinguishedName can be exported to a byte array or a string with the methods

byte[] getBytes()
String toString()

A new DistinguishedName is created from byte or string representation with one of the constructors

DistinguishedName(byte[] data)
DistinguishedName(String data)

A.1.2 UserIdentity

The UserIdentity object represents a user that has logged into the system. It contains both a user-friendly name of the user (the login name), the DistinguishedName (one or several), and the login key that is used to protect the user registration data and private keys.

Constructor for the UserIdentity object:

UserIdentity(String userName, DistinguishedName DN, SymmetricKey loginKey)

This is the basic constructor for a UserIdentity. It creates a new UserIdentity with only one DistinguishedName.

UserIdentity(String userName, Enumeration DNs, SymmetricKey loginKey)

This constructor takes several DistinguishedName:s as an Enumeration, and sets the first one as the default.

Methods for accessing the components:
DistinguishedName getDN()

This retrieves the default DistinguishedName. To retrieve all the available DistinguishedName:s, the method

Enumeration getDNs()

is used. The login key can be obtained with:

SymmetricKey getLoginKey()

and the user friendly name with

String getUserName()

A.1.3 AuthenticationObject

This object contains data needed to verify the local login. It also contains data about the policy of the local login, like for example minimum password length or the date after which the password must be changed. It returns a symmetric key that can be used to protect the PrivateKey:s of the user and other sensitive data.

When a user logs in locally, he/she has to supply an identification and some form of verification. Once the user has entered the login data, the login system uses the AuthenticationObject that belongs to that particular user to verify the authenticity of the user. The result of this verification is returned to the Login object.

The form of authentication can be classical name/password control, where the user identifies herself with a nickname and proves her identity with a password known only to the user. This method is the one that is most widely used right now, even if it has the weakness that passwords can be easy to guess [Morris 1979, Abadi 1997]. In the case of using smart cards, the identification is the smart card itself, while the authentication information is the PIN code used to unlock the card. Other more advanced implementations can be the use of biometric techniques, like face recognition and/or fingerprints to identify and authenticate the user [Huopio 1998, Davies 1994].

For any of these implementations, the authentication data obtained in the login process, whether it is a password, a PIN code or biometric parameters, is passed to the AuthenticationObject where it is verified. The AuthenticationObject stores the information needed to verify the login in a
file. This file is sensitive and therefore should be protected. In case of smart cards, for example, the authentication file is stored inside the card, and is not accessible from the outside [PKCS15]. In other systems, if the authentication file has to be stored in an unprotected area, it can use one-way functions so that it can verify the authenticity of a request, without containing the secret data itself.

The AuthenticationObject controls the access to the rest of the user’s sensitive data, like the user’s PrivateKey:s for example. How this is done depends on the implementation. In a trusted hardware device, like a smart card, the AuthenticationObject gives access rights to the area where the sensitive data is stored. These access rights are enforced by the device. If all data is stored on a computer, the AuthenticationObject contains a symmetric key (which we call login key) that is used by the system to encrypt all other sensitive user data. This login key itself is protected by encryption, using the user authenticity information as a key. After a successful login, the login key is decrypted and made available, so that the user’s private keys can be decrypted and used. The advantage of using this approach instead of encrypting the user private keys with the user password is twofold. For the first, it is more flexible, allowing for authentication systems that have other means of authentication then a password. For the second, if the password is changed, there is no need to re-encrypt all private keys and sensitive data that had been encrypted with it. The only thing that needs to be done is to decrypt the login key with the old password and encrypt it anew with the new one.

![AuthenticationObject Diagram](image.png)

**Figure A.1** : Usage of AuthenticationObject

The basic constructor is

```java
AuthenticationObject()
```
which creates a new AuthenticationObject that can be used for subsequent verifications.

The syntax of different AuthenticationObject:s differs depending on the type of authentication they are to be used for. For example, after the object has been created, the login in a password-based authentication scheme can be verified by calling the method

```java
bool verifyLogin(String userName, String password) throws AuthenticationException, OldPasswordException
```

If the login verification is correct, the method returns true. If the login verification fails, then an AuthenticationException is thrown and the method returns false. If the login is correct, but the password should be changed, an OldPasswordException is thrown and the method returns true. These exceptions should be intercepted by the Login module, which in the case of OldPasswordException should ask the user to change to a new password.

Instead of creating an empty object and then verifying the login, the AuthenticationObject can be created directly with the login parameters. Therefore, the constructor for password-based authentication is:

```java
bool AuthenticationObject(String UserName, String password) throws AuthenticationException, OldPasswordException
```

This initializes an AuthenticationObject and verifies the authenticity of the login at the same time. Otherwise, it functions in the same way as a call to the verifyLogin method. If the login has been successful, the login key can be obtained by calling

```java
SymmetricKey getLoginKey();
```

The AuthenticationObject is responsible for storing local authentication information, so it also contains methods for the administration of this information, like adding new users and removing existing ones. Which users have the right to add or remove users is specified in the policy file of the system. The method:

```java
bool addNewUser(String userName)
```

adds a new user without specifying user password. At the first login, the user is required to change the password. If the userName already exists, then the method returns false and nothing is changed.
bool removeUser(String userName)

removes a user. Returns true if successful, false otherwise.

changePassword(String passwd1, String passwd2) throws
PasswordNotMatchException, PasswordNotGoodException

changes the password of a user. This method works only if the user has logged in
successfully. If the two passwords do not match, or if they do not meet the policy
requirements (for example, if they are too short) then a
PasswordNotMatchException, respectively the
PasswordNotGoodException is thrown.

The constraints for a password (i.e. minimum length, how often it is to be
changed) are specified in the policy file of the system. This file contains sensitive
information and should be protected, either by storing it in a safe place (on a
smart card for example), by signing it, or both.

### A.1.4 Login object

When logging into a computer, the user supplies her identification and
authenticity information (whether it is a password, a PIN code or a fingerprint) to
the Login object. In its turn, the Login object takes the authenticity information
supplied by the user and sends it to the AuthenticationObject for
verification.

If the verification does not succeed, the Login object prompts the user for
another try. If the AuthenticationObject triggers the error
OldPasswordException, the Login object asks the user to give a new
password and then tries to change it by calling the changePassword method in
AuthenticationObject.

If the verification succeeds, Login reads the login key from the
AuthenticationObject. It then creates a UserRegistration and a
UserIdentity object for the user.
Other objects can then request the UserRegistration or UserIdentity objects from Login. This is done by calling the methods:

UserRegistration getUserRegistration()

UserIdentity getUserIdentity()

If the user has not successfully logged in, both these methods will return null objects.

**A.1.5 UserRegistration Object**

UserRegistration object contains all the data about the user that might be necessary. For example, it may contain the address of the user and other registration data which are not attributes of the DistinguishedName.

UserRegistration also contains the DistinguishedName:s of the user. One of them is marked as the default DistinguishedName and is used when no other one is specifically requested.

All this is stored in the user registration file. This file may be stored on the local disk, a network server, or a smart card, depending on the implementation. Some parts of it need to be protected during storage, so they are encrypted with the login key. The actual structure and content of the registration file depends on the particular implementation, and a special subclass of the UserRegistration needs to be created for it.
The constructor for the UserRegistration takes the user name as an argument. If data in the registration file has been stored in a protected form, the login key is needed in order to be instantiated:

```java
UserRegistration(String userName) throws RegistrationProtectedException
UserRegistration(String userName, SymmetricKey loginKey) throws RegistrationProtectedException
```

These two constructors instantiate the UserRegistration object, which goes on and tries to read the registration information from the file that belongs to userName. If the file is protected and cannot be accessed, the RegistrationProtectedException is thrown.

Methods that can be used to access the registration data are:

```java
String getAddress()
String getEmail()
```

Other methods can be implemented in subclasses of UserRegistration as necessary, depending on the data that needs to be stored.

**A.2 Cryptographic Key Objects**

**A.2.1 AsymmetricKeyPair**

An AsymmetricKeyPair object consists of a PublicKey and its corresponding PrivateKey. By using the constructor

```java
AsymmetricKeyPair()
```

a new key pair is generated. The keys can be retrieved with the methods

```java
PrivateKey getPrivateKey()
PublicKey getPublicKey()
```

The key pair, once created, can be stored in the system key database by calling the method

```java
bool store() or
bool store(SymmetricKey protect_key)
```
where protect_key is the symmetric key used to encrypt the PrivateKey. The methods return true, if the key pair was successfully stored.

It should be noted that PublicKey and PrivateKey objects cannot be stored separately, but only linked, as part of an AsymmetricKeyPair. The reason for that is that the system uses a PublicKey as an identification for PrivateKey:s, and otherwise, asymmetric private keys without their complementary public key have no use. Another way to store a PublicKey in the database without storing it together with its corresponding PrivateKey is to store it as part of a Certificate object. This could be the case when the PublicKey is not created locally, but received from somebody else.

A.2.2 PublicKey

This object contains an asymmetric public key. Its usage is to encrypt and decrypt data, and to verify signatures. This is done by the following methods:

```java
byte[] encrypt(byte[] data)
byte[] decrypt(byte[] data)
bool verify(byte[] signature, byte[] data)
```

The key can be exported to a byte array for transport (for example, when it is necessary to send it over the network for signature verification) by calling the method:

```java
byte[] getBytes()
```

The byte array obtained from this method can later be used to recreate the key at another location by using the constructor:

```java
PublicKey(byte[] key_data)
```

Note that here the object itself is not serialized, but only the data it contains is being sent. Therefore, when instantiating it on another virtual machine, the PublicKey object created contains the original data, but the object itself, with all its built-in logic, is from the local environment, and can therefore be trusted in its functionality. So verifying a signature with a key that has been received from the network can be done, as it is not the logic of the key that has been imported, but only the key data.

A.2.3 PrivateKey

It contains an asymmetric private key. Its usage is to encrypt, decrypt and sign data. A PrivateKey object can be retrieved from the system database by specifying its corresponding PublicKey and the UserIdentity object of the
user. UserIdentity object is used to open the access to the key. If the key is stored on the local disk, then the login key from the UserIdentity object is used to decrypt it. If the key is stored inside a secure hardware device, the UserIdentity is needed to gain access rights to it. The syntax of the constructor is:

PrivateKey(PublicKey pk, UserIdentity uid)

PrivateKey can encrypt and decrypt by using the methods:

byte[] encrypt(byte[] data)
byte[] decrypt(byte[] data)

Another usage is to sign data. A signature is created by calculating a hash of the data and then encrypting it. The result is a byte array:

byte[] sign(byte[] data)

The data that the PrivateKey consists of can be exported to an encrypted byte array using the method:

byte[] getProtected(SymmetricKey prot_key)

Here all parameters that are part of the PrivateKey are first encoded into a byte array, then this array is encrypted with prot_key. A new PrivateKey can be created from that byte array with the constructor

PrivateKey(byte[] key_data, SymmetricKey prot_key)

As in the case of the PublicKey, the PrivateKey object is not itself serialized. Only the data inside is exported. When it is reinstated, a new PrivateKey object is created by the system, and then the private key parameters are inserted. Therefore, a PrivateKey object can be trusted to function correctly, even if the key data has been received from an unsecure location.

A.2.4 SymmetricKey

Also called secret key, this key is used for encrypting and decrypting data with a symmetric algorithm. The main advantage of a symmetric algorithm compared to an asymmetric one is that symmetric encryption and decryption are generally an order of magnitude faster on comparable implementations [Eberle 1993, Brickell 1990]. Therefore, bulk data is usually encrypted with a symmetric algorithm.

A new key is created by calling the constructor:
SymmetricKey() or
SymmetricKey(BigInteger seed)

The latter constructor is used when specifying a seed for the random number
generator is desired. Another way of creating a key, that is sometimes necessary,
is to generate it from a human-readable password. The constructor used for this is:

SymmetricKey(String password)

Encryption and decryption is done by:

byte[] encrypt(byte[] data)
byte[] decrypt(byte[] data)

SymmetricKey:s are, contrary to asymmetric PublicKey:s, secret. If the SymmetricKey is revealed, all the messages that have been protected with it can be immediately decrypted by its interceptors. Therefore, prior to sending it over a network, a SymmetricKey needs to be itself protected. This is done by encrypting it with the public key of the receiver, so that it can be later decrypted with the corresponding private key. The method for doing that is

EncapsulatedObject getProtected(PublicKey peer_puk,
PrivateKey user_prk)

The getProtected method not only protects the key, but also signs it with the user private key. The result is an EncapsulatedObject. This method really is just creating a new EncapsulatedObject for the key.

Another method for the protection of the key is to protect the key for storage, with another SymmetricKey or with a password. Protection with a SymmetricKey is done with:

byte[] getProtected(SymmetricKey prot_key)

where the prot_key can be for example the login key from the UserIdentity object. For protecting the key with a password, the method is

byte[] getProtected(String password)

Here a SymmetricKey is first generated from the password using the

SymmetricKey(String password)
constructor, and then it is used to protect the key. The byte array obtained from these `getProtected` methods can be transformed again into a `SymmetricKey` object with the constructors

```java
SymmetricKey(byte[] data, SymmetricKey prot_key) or
SymmetricKey(byte[] data, String password)
```

# A.3 Certification Objects

## A.3.1 Certificate

This object contains the certificate of a user, server or program, according to the X.509 standard [X509]. It is created by a Certification Authority (CA) server, subsequent to a `CertificationRequest` object sent from a `CertificationClient`.

A Certificate, among other data, contains a `PublicKey`, a `DistinguishedName` associated with that public key and the signature of the CA that certifies this association. It may also contain the `Certificate` or the `CertificateChain` of that CA and other data related to the user, which is not contained in the `DistinguishedName`.

When receiving a certificate from somebody, the first thing to do is to verify it, that is to check that the signature that certifies the association between the `PublicKey` and the `DistinguishedName` is correct. For this, the `Certificate` of the CA is used. That `Certificate` must be verified in its turn, and so on until the certificate of a known and trusted authority is found. This verification can be done in two ways. The first way is to verify only the validity of the signature, without verifying the certificate of the signing authority. This can be done for example when the signing authority is known to the receiver. The method used for this is

```java
bool verify()
```

and it returns true in case the signature is correct. The other way of verifying the `Certificate` is to submit it to the `CertificationClient` object, which in its turn verifies each certificate in the chain until it finds a certificate that it trusts, and also checks whether a certificate is not revoked in a Certificate Revocation List (CRL).

Once verified, a `Certificate` can be stored in the system database. This is done by calling the method
bool store()

which returns true if storing was successful. Once stored in the database, the Certificate can be at any time retrieved and reinstantiated. The constructors used for retrieving existing certificates are

Certificate(DistinguishedName DN) or
Certificate(DistinguishedName DN, String usage) or
Certificate(PublicKey puk)

The first constructor retrieves the certificate that has the specified DistinguishedName. In the case several Certificate objects exist with the same DistinguishedName, the usage of the public key can be used for deciding which one to be used. The third constructor retrieves the Certificate that contains a certain PublicKey object.

Certificate has methods for accessing all items it contains. The DistinguishedName of the owner is retrieved by

DistinguishedName getDN()

The PublicKey of the owner is obtained by calling

PublicKey getPublicKey()

The Certificate of the signer is obtained by

Certificate getSignerCertificate()

and the CertificateChain of the Signer is obtained by

CertificateChain getSignerCertificates()

The data of the Certificate object can be exported to a byte array for transportation. The method used is

byte[] getBytes()

A new Certificate object can then be created from such a byte array by calling the constructor

Certificate(byte[] data)
As with the PublicKey and PrivateKey objects, this approach of exporting the data of a certificate to a byte array and then creating a new certificate with that data makes it possible to fully trust the verify() method inside the Certificate object.

A.3.2 CertificateChain

A CertificateChain object consists of several Certificate:s that certify each other. It is used to prove that a certain Certificate is certified by CA:s all the way to the top of the hierarchy.

An empty CertificateChain object is created by the constructor

CertificateChain()

Certificate objects are inserted in the chain using the method

addCertificate(Certificate cert)

The CertificateChain object then automatically inserts the new Certificate in the correct place in the chain. All the Certificate:s can be retrieved using:

Enumeration getCertificates()
A.3.3 CertificateRequest

To obtain a new Certificate, a client has to create a CertificateRequest and send it to the CA. The CA can then verify the request and create a new certificate for the client. A CertificateRequest has to contain the PublicKey and the DistinguishedName of the client according to the PKCS#10 standard [PKCS10]. The syntax for its constructor is therefore:

CertificateRequest(PublicKey puk, DistinguishedName dn)

The methods to retrieve this data from the object are:

PublicKey getPublicKey()
DistinguishedName getDN()

For transmission, a CertificateRequest can be exported to a byte array:

byte[] getBytes()

The constructor that recreates a CertificateRequest object from the byte array is:

CertificateRequest(byte[] data)

A.3.4 CertificationClient

The CertificationClient object is the part of the user local system that takes care of all communications with the certification infrastructure. It can send a certificate request and receive the response from the CA, or it can verify a Certificate object, checking its credentials and also checking that it is not revoked in a CRL [RFC2459]. The CertificationClient is started with the constructor

bool CertificationClient()

which returns true if everything works, and false if the CertificationClient could not contact a CA. Once started, Certificates can be submitted to be checked:

bool verify(Certificate cert)

To create a new Certificate, the CertificationClient takes all data necessary and creates a CertificateRequest object, which is then sent to the default CA:
bool sendCertificateRequest(PublicKey puk, DistinguishedName dn) or
bool sendCertificateRequest(byte[] data)

The CA can then be asked if the Certificate has been issued or not. The method for doing that is

bool checkCertificate()

If the CA has issued the Certificate, it is fetched and stored in the system database by the CertificationClient.

The name and address of the default CA, as well as other parameters necessary to the CertificationClient are stored in a file, called Certification Policy file. The CertificationClient needs also to store the certificate of a known and trusted Certification Authority. This certificate is also stored in a file on the disk or inside a smart card.

A.3.5 CRL

CRL stands for Certificate Revocation List and it contains the numbers of all certificates that have been revoked before their expiration date together with a reason for their revocation [RFC2459]. CRL:s are received by the CertificationClient from the CA:s.

An empty CRL is created by the constructor
CRL()

Certificates to be revoked are added to it with the method

bool addCertificate(Integer serialNumber, String reason) or
bool addCertificate(Certificate cert, String reason)

The **CRL** can be exported to a byte array in a standard format with the method:

byte[] getBytes()

and then be recreated from that byte array with the constructor

CRL(byte[])

The method

bool check(Certificate cert)

checks whether **cert** is in the **CRL**. If it is, it should be discarded.

**A.4 Secure Session Objects**

**A.4.1 EncapsulatedObject**

An **EncapsulatedObject** contains a normal **Java Object**, which it encrypts and signs with the asymmetric keys that are supplied to it. The **EncapsulatedObject** can be transformed into a byte array that can be sent over the network and then used to create a new **EncapsulatedObject**. The object that is contained in the **EncapsulatedObject** has to be serializable.

Here are some constructors for the **EncapsulatedObject**:

EncapsulatedObject(Object message)
EncapsulatedObject(Object message, PublicKey peer_puk)
EncapsulatedObject(Object message, PublicKey peer_puk, PrivateKey user_prk)

The first constructor creates an **EncapsulatedObject**, but does not specify any receiver for it. The second adds the public key of the receiver, so that the object can be encrypted, but not signed. The third constructor adds both the
public key of the receiver and the private key of the sender, thus both encrypting and signing the contents. Other constructors can contain a Certificate of the peer instead of the public key, or a Certificate of the sender and the UserIdentity of the sender instead of the sender’s PrivateKey:

```
EncapsulatedObject( Object message,
       Certificate peer_cert,
       Certificate user_cert,
       UserIdentity user_id)
```

Once the EncapsulatedObject is created, if it contains at least the public key of the receiver, it can be used to encrypt the data and return the encrypted object as a byte array, using this method:

```
byte[] getProtected() or
byte[] getProtected(String encoding_id)
```

Depending whether the user PrivateKey has been given or not, the resulting byte array will contain the object encrypted and signed, or only encrypted. In the second method it is possible to specify the encoding standard to be used for the creation of the byte array. For example, one standard that can be used is PKCS#7 [PKCS7].

When the getProtected method is called, the Object inside the EncapsulatedObject is serialized, encrypted with a symmetric key which is automatically generated and possibly signed with the private key of the sender. The symmetric key used to encrypt the serialized Object is in its turn encrypted with the public key of the receiver and the result is encoded in a byte array, according to the standard specified by the encoding_id. Before encrypting the object, other data may be added to the EncapsulatedObject. For example, several receivers can be added, using the method:

```
bool addReceiver(PublicKey peer_puk) or
bool addReceiver(Certificate peer_cert)
```

The results of adding several receivers is that the object is made readable for all of them. This is done by making several copies of the SymmetricKey used to encrypt the bulk data, and encrypting these copies, each with the public key of a new receiver. These separate encrypted symmetric keys will then be added to the package, making it possible for all receivers to obtain the original symmetric key and decrypt the contents of the message.
In the same way, several signatures can be added to an `EncapsulatedObject`. This can be useful when a message with multiple senders is sent and each sender needs to sign it. The method used for this is

```java
bool addSigner(PrivateKey user_prk) or
bool addSigner(Certificate user_cert, UserIdentity user_id)
```

In the case of the second method, the `PrivateKey` that corresponds to the `user_cert` is automatically fetched from the system database.

![EncapsulatedObject structure](image)

**Figure A.5**: EncapsulatedObject structure

Once the protected byte array has been created, it can be sent over the network or stored on a local disk. A new `EncapsulatedObject` can later be created from it. The syntax used is:

```java
EncapsulatedObject(byte[] in_data, PrivateKey user_prk) or
EncapsulatedObject(byte[] in_data,
                  Certificate user_cert,
                  UserIdentity user_id)
```

In the second method, the `PrivateKey` of the user, necessary to decrypt the message, is fetched automatically from the database. If any step in the decryption of the message or the verification of its signature fails, the returned object is null.

Once an object is successfully created, the Java `Object` it contains can be retrieved by the method

```java
Object getContents()
```

The `Certificate` of the signer can be obtained by calling
Certificate getSigner()

This resulting Certificate can now be verified by the CertificationClient.

**A.4.2 SecureSession**

This object opens a secure communications session with another entity on the network. Its main function is to perform a strong authentication protocol between the two entities and then open a secure channel between them. The object is initialized, then it produces messages that are to be sent between the parties. The messages received are at their turn given to the object, which can then check if the process is going on correctly, and if it is, create the next message.

*SecureSession* needs the following items: the Certificate of the local user, the Certificate of the peer, and the PrivateKey of the local user. It is created by the constructor:

```
SecureSession (Certificate user_cert,
               PrivateKey user_prk,
               Certificate peer_cert)
```

The messages are obtained by calling the method

```
byte[] getNext()
```

The message received from the peer entity is given to the *SecureSession* object using the method:

```
receive(byte[] message)
```

The result of the authentication process, if it has been successful, is a SymmetricKey object (session key) that can be used for further secure communication between the parties. The method

```
bool authenticationSucceeded()
```

returns true if the authentication was successful and false otherwise. The SymmetricKey that results from the authentication can be obtained by calling the method

```
SymmetricKey getKey()
```
If the authentication fails, the exception `AuthenticationException` is triggered.

### A.5 Access Control and Authorization Objects

#### A.5.1 Capability
This object represents an action that is allowed to its owner. It can be for example a movie ticket, granting the entrance of its owner to the theater, or it may allow access to a system resource. The `Capability` itself does not contain any information about its owner, neither is it protected or authenticated in any way.

A `Capability` consists mainly of a name that identifies it and a description. It may be subclassed to contain other data as well, depending on its intended usage. For example, the movie ticket could be called just “Movie Ticket”, its description may be “Virtual piece of paper granting entrance to the theater” and it would contain, as additional parameters, the starting time of the movie, its name and the number of the seat.

A new `Capability` is created with the constructor

```java
Capability(String name, String description)
```

The data it contains can be accessed by

```java
String getName()
String getDescription()
```

In the case of the movie ticket, other methods may be

```java
String getFilmName()
String getStartTime()
int getSeatNumber()
```

#### A.5.2 Attribute Certificate
This object represents an authenticated `Capability`. It contains a `Capability` object that is signed by its issuer. It may also contain the name of its owner, if that is needed (some authorizations may be anonymous) and the `DistinguishedName` of the issuer.
A new AttributeCertificate is created by its issuer (an entity with the authority to issue it) using one of the constructors:

```java
AttributeCertificate ( Capability cap,  
    Certificate issuer_cert,  
    UserIdentity issuer_id)
AttributeCertificate ( Capability cap,  
    Certificate issuer_cert,  
    UserIdentity issuer_id,  
    Certificate owner_cert)
```

The PrivateKey of the issuer, used for signing the Capability, is fetched from the system key database.

The AttributeCertificate can be exported to a byte array with:

```java
byte[] getBytes()
```

This byte array contains the serialized Capability, the signature and the Certificates of the issuer and of the owner. It can be transformed back into an AttributeCertificate object with:

```java
AttributeCertificate (byte[] data)
```

The AttributeCertificate can then be verified with the method

```java
bool verify()
```

which checks that the PublicKey in the issuer_cert verifies the signature on the Capability. The data of the AttributeCertificate, that is the Capability and the Certificate:s of its issuer and owner, can be obtained by:
Capability getCapability()
Certificate getIssuer()
Certificate getOwner()

A.6 Smart Card Objects

A.6.1 SmartCard Object
Methods of this class provide functions for activating, initializing, and managing smart card contents.

Initialize and personalize smart card:

    void initialize(SmartCardID, SOPIN)
    void personalize(UserPIN, Non-repPIN)
    SmartCardID getInfo()

Method `initialize()` is used to create the initial (system) files in the smart card. These files are manufacturer dependent, so before performing this method, the ATR must be obtained and analyzed in order to recognize the manufacturer and the type of the smart card. Method `personalize()` is used to define the user PIN:s on the card. Finally, `getInfo()` can be used to list the content of the DIR directory.

Manage user authentication:

    void changePIN(PINType, oldPIN, newPIN)

Method `changePIN()` can be used to change one of the PINs in the smart card.

Activate and close smart card:

    void open()
    void close()

Method `open()` is used to open the smart cards, while method `close()` is used to close smart card.

A.6.2 Terminal Object
Methods of this class provide smart card reader devices management and access to the smart cards functions.
Establish and close terminal sessions:

```java
void open()
void close()
```

Method `open()` is used to open the session with the smart cards terminal. Method `close()` is used to close the opened session.

Obtain reference or information about inserted card(s):

```java
SmartCard getSmartcard(SmartCardID)
SmartCard waitForSmartCard(SmartCardID, Timeout)
```

Method `getSmartCard()` can be used to obtain information about the inserted smart card. Method `waitForSmartCard()` is used to wait until ATR signal is returned from the smart card.

**A.6.3 SCSession Object**

Methods of this class provide functions to manage applications or objects on a smart card via secure or regular smart card sessions. Session is a logical combination (collection) of smart card applications that a user, who opened the smart card, may use. Sessions may be dynamically opened and closed, i.e. the user may activate different combinations of smart card applications, depending on user needs and the IT environment.

Session activation and management – These methods associate SC applications within the card with the current session:

```java
SCSession open(SessionProfile)
SCSession open(SmartCard, PINType, PIN)
SCSessionID getInfo()
void close()
```

Application management:

```java
SCApplication[] getApplications()
SCApplication getApplication(SCApplicationID)
void removeApplication(SCApplicationID)
```

**A.6.4 SCApplication Object**

The methods of this class provide the possibility to create, update, delete, and activate smart card applications.
Manage smart card applications:

void create (SCSession, SCAplicationID)
void remove (SCSession, SCAplicationID)
SCApplication getApplication(SCSession, SCAplicationID)
SCApplication[] getApplications(SCSession)
SCApplicationID getInfo()
References


[JAAS] Sun Microsystems, Inc., Java Authentication and Authorization Service,  


[Lee 2002] Seungjoon Lee, Bohyung Han, Minho Shin, “Robust Routing in Wireless Ad Hoc Networks”, 2002 International Conference on Parallel Processing Workshops (ICPPW’02) August 18 - 21, 2002 Vancouver, B.C., Canada


Nicolas Prigent, Christophe Bidan, Jean-Pierre Andraeux, and Olivier Heen, “Secure Long Term Communities In Ad Hoc Networks”, 2003 ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN ’03) October 31, 2003 George W. Johnson Center at George Mason University, Fairfax, VA, USA.

[RFC2459] Request for Comments, RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile


[X500] CCITT X.500 Series Recommendations, The Directory - overview of concepts, models and services, December 1988


