Managing Privacy in Peer-to-Peer Distribution of Clinical Documents

by

Christina Obry
Bachelor of Computer Science, University of Paderborn, Germany 2003

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Managing Privacy in Peer-to-Peer Distribution of Clinical Documents

by

Christina Obry

Supervisory Committee

Dr. Jens Weber-Jahnke, Supervisor, Department of Computer Science

---

Dr. Hausi A. Müller, Department Member, Department of Computer Science

---

Dr. Alex Thomo, Department Member, Department of Computer Science

---

Dr. Benjamin Jung, External Examiner (Department of Health Information Science, University of Victoria)

Supervisory Committee:

Dr. Jens Weber-Jahnke, Supervisor, Department of Computer Science

---

Dr. Hausi A. Müller, Department Member, Department of Computer Science

---

Dr. Alex Thomo, Department Member, Department of Computer Science

---

Dr. Benjamin Jung, External Examiner (Department of Health Information Science, University of Victoria)

ABSTRACT

Security and privacy are two of the most important aspects of any medical information mediation system. Governments have established privacy legislations to prevent abuse of patients' personal data. These legislations require organizations to obtain consents prior to information usage and exchange. The consents are defined as policies.

However, policies are often not precise and adequate enough to address all possible eventualities and exceptions. Unanticipated emergency cases may cause conflicts between a patient's right for privacy and the need to receive treatments from well-informed caregivers. In these situations, the patient's safety should have precedence. Therefore, caregivers should have the ability to override the patient's privacy policies on behalf of the patient.

This thesis presents a mechanism, which restricts access to sensitive, medical data based on defined policies, but which also allows overriding the policies in emergency cases. The overriding process is monitored and audited in order to prevent misuse.

Table of Contents

List of Tables

List of Figures

Acknowledgement

I like to express my deepest gratitude to Dr. Jens Weber-Jahnke, my supervisor and friend, without whom this research would not have been possible. Thank you Jens for your guidance, advice and support during my studies.

Thanks to my friends and colleagues in the Pervasive Primary Care Informatics Lab, who have made this research a very pleasant journey. Especially, I like to thank Adeniyi Onabajo for always taking the time to discuss and review my work, for having faith in me when I did not and for reminding me that sometimes you have to take baby steps in order to reach a goal. You are definitively the best "co-supervisor" and friend one could wish for. I also like to thank Glen McCallum for many, often non-work related talks and for his "This machine is for THESIS USE ONLY!" sign, which somehow helped me to focus on the thesis during the last few months. I will surely miss the fun times in the lab.

Many warm thanks to my family for their endless love, encouragement and for proving that you can be close even if you are far away.

Finally, thanks to all my friends who have been supportive over the years.

Dedication

Für meine Eltern, Gudrun und Wolfgang Obry

(To my Parents, Gudrun and Wolfgang Obry)

Chapter 1

Introduction

Knowledge is one of the most valuable assets of many organizations. Hence, enabling information exchange and knowledge sharing among collaborating organizations has become crucial. Additionally, information system integration is often seen as a solution for lowering costs of providing services while improving their quality and efficiency. Whenever personal information is exchanged there are always questions on security and privacy. To protect individuals against abuse of their personal information, governments have established privacy legislations such as the Personal Information Protection and Electronic Documents Act (PIPEDA) [24] and the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [11]. These legislations require organizations to get the consent of individuals prior to collecting and using their information. Furthermore, organizations have to be responsible for protecting such information, in particular, if it is exchanged with third parties.

Personal medical data for example is extraordinarily sensitive. That is why individual consents are important in handling personal health information. Patients should be able to grant or deny access to their health records, especially, if this information may be transmitted to other organizations. Thus the design and implementation of security mechanisms for distributed health networks should consider legislation, organizational policies and patient consents as essential requirements.

However, it is not possible to specify policies precise and complete enough to address all possible eventualities and exceptions [36]. Unanticipated emergency situations may oc-

cur and may require access to medical data, which according to the policies is not granted. Therefore, there is a need to be able to override policies on a need-to-know basis.

Chapter 2

Problem Description and Hypothesis

Paper-based medical records can be incomplete, fragmented (different parts in different locations), hard to read and (sometimes) hard to find. Therefore, health organizations are striving to move from paper-based records to electronic health information management systems. Many health care institutions already employ information systems to manage certain aspects of patient care. However, these systems are often disjointed. In order to communicate with other clinicians, often the information is printed from one system and sent to the other clinicians via mail. This kind of communication can cause delayed or inappropriate patient treatment, lost or destroyed medical records, and is overall inefficient. Whereas exchanging electronic medical records would be much faster, cheaper and more efficient.

Security and privacy are important requirements for any paper-based or computer-based system handling personal medical information. Many voices have raised concerns that a move toward computer-based handling and distribution of personal health information may erode individual privacy. Others state that privacy can in fact be better protected in a computer-based system, if it is designed in the right manner [5].

Our research focuses on developing a mechanism that enables secure peer-to-peer distribution of electronic medical records in Clinical Document Architecture (CDA) format. The CDA is a standardized way of representing machine-processable electronic medical information, and it was developed by the international standards body Health Level 7 (HL7).

In order to ensure security and privacy, policies are used to determine which informa-

tion is allowed to be exchanged and which information is not. However, computer-based policies have the disadvantage that they are too strict and inflexible [14], since they are made with the assumption that all access permissions are known at the time when the policy is created, but in practical scenarios unanticipated situations can occur and therewith it may be that a caregiver needs to have access to certain information in order to provide better care even if according to the privacy policies the access is not granted. In emergency cases like this it should be possible to override a computer-interpreted privacy policy in order to guarantee the best treatment for the patient. However, such a solution should always be monitored and reported, in order to prevent misuse as well as for auditing purposes.

Security and confidentiality are important especially in the health domain and enforcing them strictly is one way of preventing the abuse of sensitive information. However, in practice a more flexible approach is needed. We believe that a medical mediation system that:

- restricts access to particular sensitive information based on patients' and clinicians' consent as well as legislative requirements
- ensures that such restrictions can be overridden on a need-to-know basis only
- monitors, audits and reports the overriding of policies

will on one hand fulfil the need for security and privacy and on the other hand allow enough flexility to be used in practice.

Chapter 3

Approach Overview

In close collaboration with researchers in the School of Health Information Science at the University of Victoria as well as the Department of Family Practice at the University of British Columbia, the Pervasive Primary Care Informatics Laboratory is developing a policy-based security mechanism to enable secure information sharing within a peer-to-peer network of health organizations and clinicians. Our research work has been focusing on developing mediation technology to facilitate information exchange among heterogeneous medical information systems [34, 7].

For policy-based security mechanism our general approach is to provide an adapter component, called CDAShip, which allows different organizations and clinicians to participate by abstracting their information system's internal implementations and structures and exposing appropriate data (as determined by the individual organizations). This follows Canada Health Infoway[1]s Electronic Health Record Solution (EHRS) Blueprint specification, which defines Health Information Access Layer (HIAL), an interface specification for services, which enable interoperability, security and integration [22]. Secure transmission of information is essential for adopting distributed health information mediation solutions. Therefore, encryption of the data is required before exchanging the information. However, there is a need to extend the encryption with mechanisms to accommodate different policies including patients' and clinicians' consents as well as organizational and legisla-

[1]Canada Health Infoway is a Canadian organization, which promotes the adoption of electronic health information systems `http://www.infoway-inforoute.ca`

tive requirements. The CDAShip component uses a policy-based encryption mechanism to provide advanced security. One subcomponent of the CDAShip creates a security layer by ciphering sections of the information separately according to the policies. Furthermore, the CDAShip uses a mechanism that allows decryption of the non-authorized sections in emergency cases. However, to ensure that this step is only taken if necessary and important for the patient care, the decryption will be monitored, audited and reported.

Another important aspect of the CDAShip is that it allows caregivers with different access rights to view different information. This means that it is possible for caregivers with less access rights to distribute information that are not known to them to other caregivers with more access rights. This is done via transcription of the documents before forwarding them.

Further, the CDAShip exchanges the documents on a peer-to-peer network, in order to provide flexibility and allow caregivers from any location to participate in the information exchange. Another advantage of a peer-to-peer network over a central server is that there is no single point of failure, since it replicates data over multiple peers. Also a peer-to-peer network is easily extensible and scalable, since each peer provides the resources such as the bandwidth and computing power.

3.1 General Mechanism

Let us assume that each caregiver in our peer-to-peer network has a CDAShip installed, and that before sending clinical documents to other caregivers, the CDAShip would run a policy evaluation to determine which information the recipient caregiver is allowed to access. One way to proceed could be to cut the unauthorized sections out of the document. However, in this case the receiving clinician is not aware that there is information missing and may assume that his record is complete. This lack of knowledge could effect patient treatment negatively. Alternatively, it would be possible to replace the unauthorized sections with a note indicating that there is more information available to which the access is currently not

granted. This would ensure that caregivers are aware of the missing information. If caregivers need to access the information in order to give the patient a better treatment, then they could request the missing information from the originating caregiver. However, the information may be lost in case the original document does not exist anymore or the document holder is not available. In our approach we are encrypting the unauthorized sections. By using encryption we allow the caregivers to see that there is more information available, but we also allow them to access the information by decrypting them. The mechanism used to retrieve the ciphering key, which is needed for the decryption is explained later on in this section.

In addition to the encrypted document, the policies that were applied to the document will also be transmitted, to ensure that the receiving caregiver is able to apply the patient's policies in case the received document should later be forwarded to another caregiver.

It is further assumed that if caregivers want to forward information they have received from other caregivers in encrypted form, that the CDAShip would decrypt the encrypted sections in order to run the policy evaluation on the whole document and to encrypt the document according to the policies, which apply to the caregiver, who will receive the forwarded document. The decryption and encryption will be "invisible" to all caregivers, since both are done internally in the CDAShip. Figure 3.1 shows an example of distribution of a clinical document from patient X. The distribution is described below.

1. X's family doctor D sends X to an specialist S1 in the hospital. According to the policies S1 is not allowed to access the information about X's family history, so this section has to be encrypted.
2. S1 reports the results of the patient treatment back to D.
3. D further refers X to a different specialist S2, who is trusted and therefore can access all information.
4. S2 also sends X to the specialist S1. Again parts of the document are encrypted according to the policies.

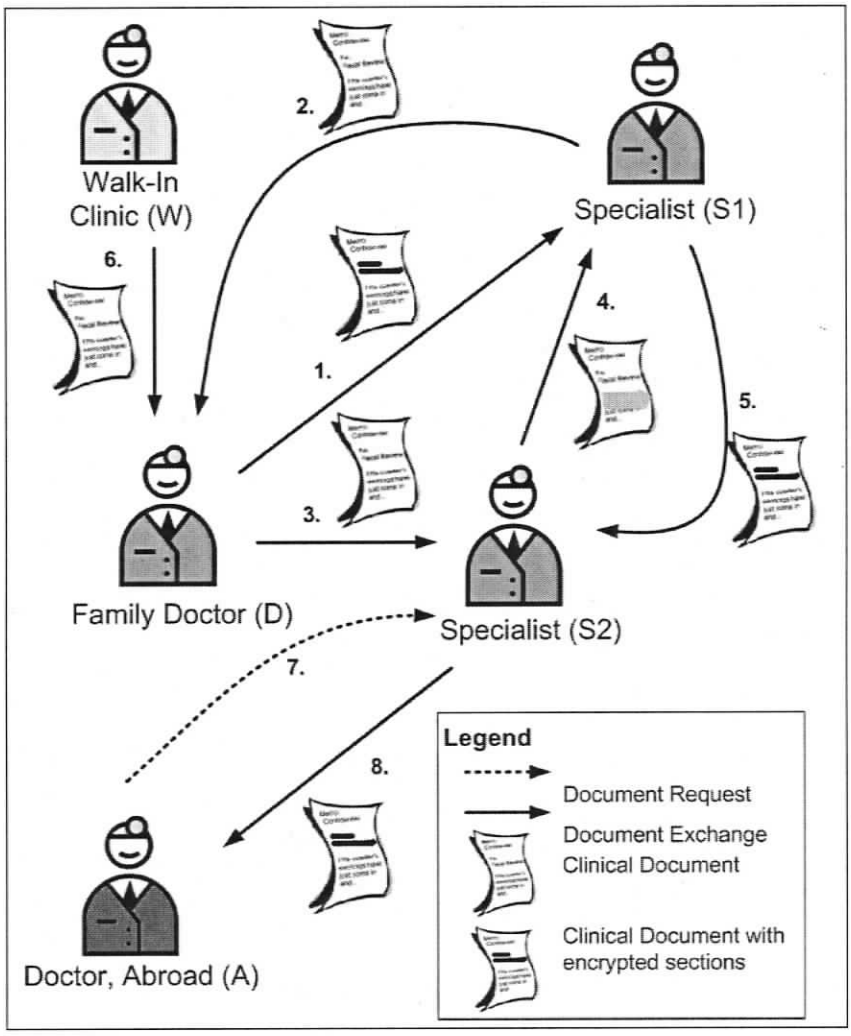

Figure 3.1. *Example P2P distribution of CDA documents*

5. S1 now sends the result to S2. S1 has already received information about X from D. However, parts of this information are still encrypted. Therefore, S1's CDAShip decrypts the encrypted sections and runs the policy evaluation on all available information. Depending on the policies, S1's CDAShip will encrypt unauthorized sections again.

6. Assuming X is visiting a walk-in clinic, at a time when his family doctor is closed. After X's visit the doctor in the walk-in clinic would send X's chart to the family

doctor. Again no encryption other than the transport level encryption is needed.

7. Let us assume X is on vacation and has a car accident. X would specify S2 as last doctor that X has visited and would authorize A to request X's medical information from S2.
8. S2 would send X's documents to A. The encrypted document, which A holds, will be decrypted and encrypted as in step 5.

This is just an example of the distributed nature of the information exchange among caregivers. In addition it shows that a network, which realizes this kind of information exchange has to be flexible, scalable and decentralized, such as a peer-to-peer network. This does not exclude the use of a central repository, which holds information of all patients of one region and which would also allow access to patient information even if the doctor's office is closed. However, such central repository is limited to a certain region and therefore is not as flexible and scalable as the practice requires. A combination of both would be possible in order to combine the flexibility that the peer-to-peer network offers and availability of a central repository. However, this is out of the scope of this thesis.

As earlier motivated, caregivers should have the ability to override the patient's privacy policies on behalf of the patient and access unauthorized information in emergency cases. Due to this fact the keys used to encrypt the unauthorized sections have to be available upon request. A solution for a key distribution mechanisms could be not to include the key. This means that the key has to be stored in the information system of the clinician, who initially sent the document. However, the clinician that is holding the key for the encrypted section may not be available when the information is needed, then the key cannot be retrieved and the important information cannot be decrypted. Alternatively, the key could be stored in a central key server, but this server could also be temporarily unavailable, which leads to the fact that the needed information is not accessible. Since the information is needed in emergency cases this is not acceptable. In our approach the key used to encrypt the unauthorized sections will also be exchanged using a secret-sharing scheme. The idea behind the secret-sharing is to divide a secret, in this case the secret key, into certain pieces

called shares. These shares and a list of who owns a share are then distributed among the caregivers. Furthermore, the pooled shares of a specific subset of caregivers are enough to reconstruct the original share. This means that the encrypted data can be accessed, if a certain number of shareholders provide their key share. The caregiver, who needs to access the unauthorized data, would request the shares from the other caregivers including the reason why the data needs to be viewed. Further, the CDAShips of all caregivers, who receive this request, would first automatically audit the request and notify the patient about the distribution of the key share and then send their key share to the requesting caregiver. This way the privacy breakage is also monitored and audited. Another advantage of this method is that not all caregivers are needed to reconstruct the key, which is important since it may be possible that not all CDAShips of the caregivers are available at the time when the key is needed. Also the key can only be reconstructed with valid and original key shares, using encryption and digital signature for the exchange of key shares ensures authenticity, integrity as well as non-repudiation. Further details about secret-sharing can be found in Section 4.4.

3.2 Context-based Ciphering Mechanism

Before a clinical document can be transmitted to other clinicians on the peer-to-peer network it has to be encrypted to ensure that only the authorized people can access the patient information. Since medical data is extraordinarily sensitive it is important to consider the patients' and clinicians' consents as well as legislative requirements before transmission. Policies are used to determine which information is permitted to be exchanged, to whom and for which reason. A policy evaluation defines which sections of the CDA document the receiving party is allowed to access and which are not. The unauthorized sections will be encrypted using a secret key, which will be shared using a secret-sharing scheme as described in Section 4.4. In order to be able to request the key shares, a list of who has received a share of the key as well as all applicable policies will be transmitted together.

Once the document sections are encrypted according to the policies and the key information and policies have been gathered, the documents digest will be created and signed using digital signature mechanism.[2] The signed message digest will be attached to the document itself. To ensure that the signature cannot be compromised during the transfer it also has to be encrypted.

The signature will be encrypted using a new secret key, which then will be further encrypted using public key encryption. This is done because public key encryption is time consuming and a secret key is generally smaller than a digital signature. The encrypted secret key used to encrypt the signature, the document, the key share information and the policies will now be transmitted to the receiver using a secure channel (see Section 4.3.1.2). Figure 3.2 illustrates the explained encryption mechanism.

In order to retrieve the clinical data the recipient will decrypt the received secret key used to encrypt the signed message digest and use this key to decrypt the signature and execute the signature verification.[3] Due to the characteristics of a digital signature the verification ensures authenticity, that is the received information is really from whom it claims to be, as well as integrity, meaning that the document has not been compromised during the data transfer. Figure 3.3 visualizes the decryption mechanism.

3.3 Non-monotonic Information Forwarding

It may be necessary for caregiver $C1$ to forward a received document to another caregiver $C2$ under the reason $R1$. In case the document includes any encrypted section that $C1$ is not allowed to access, $C1$'s CDAShip would request the key shares from the other caregivers providing the reason why the information has to be forwarded. After receiving the needed key shares, $C1$'s CDAShip reconstructs the secret key and decrypts the sensitive data. Once the document is free of any encrypted sections a new policy evaluation using

[2]For detailed description of the digital signature mechanism see Section 4.3.2.1.
[3]Digital signature verification is described in Section 4.3.2.1.

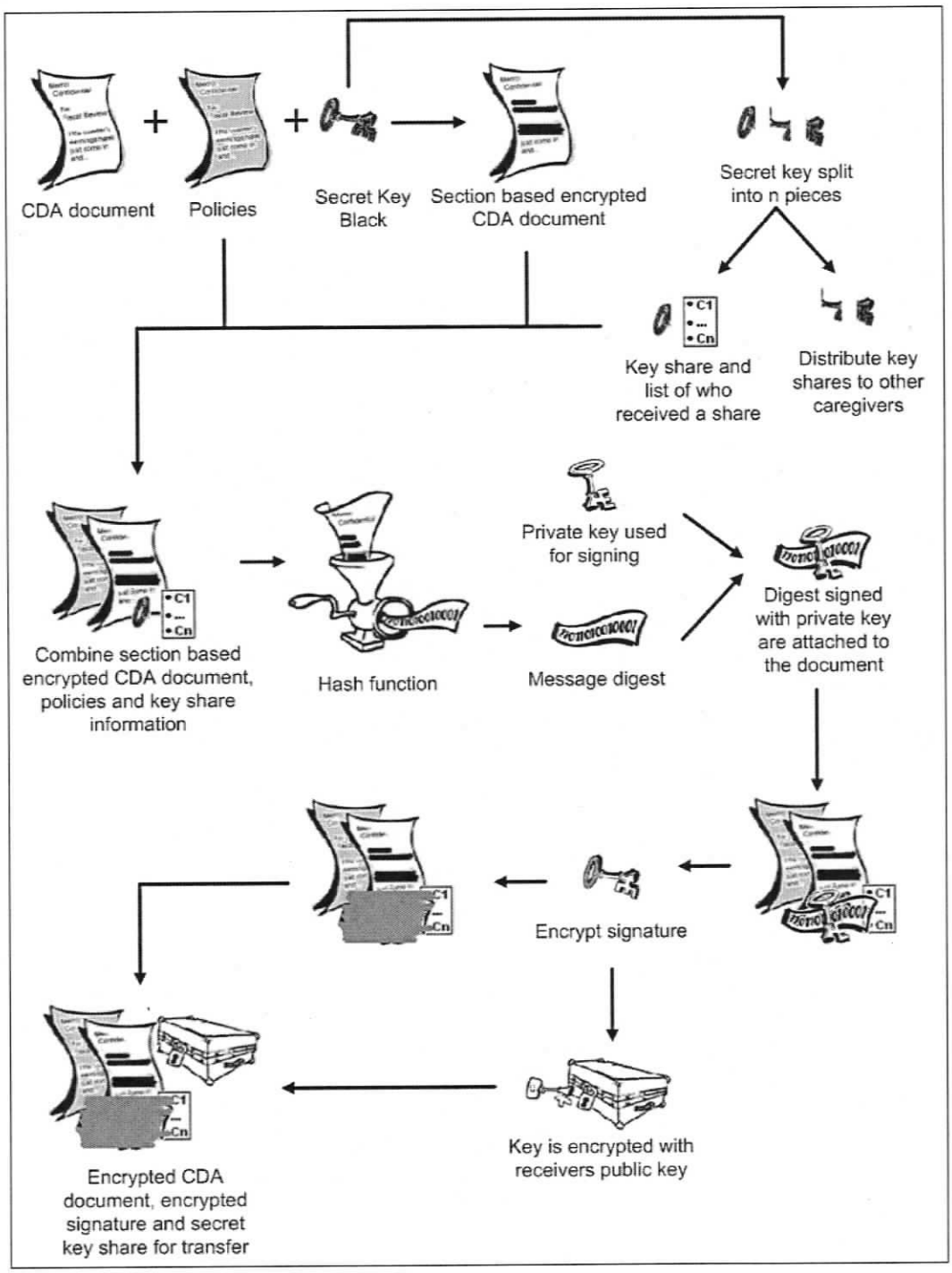

Figure 3.2. *Encryption mechanism*

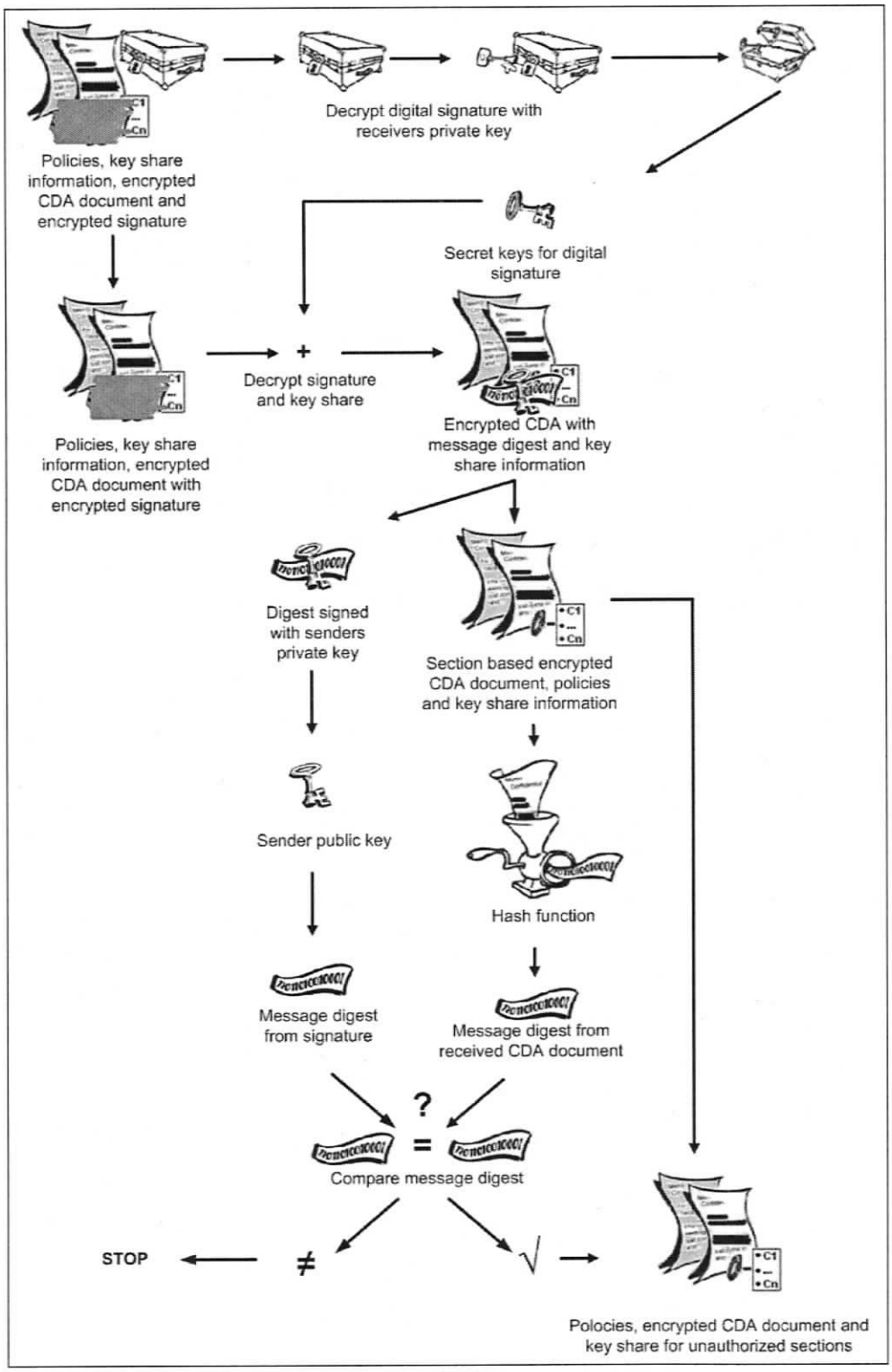

Figure 3.3. *Decryption mechanism*

the policies that apply to $C2$ will be executed and the document will be encrypted according to the policies as described in Section 3.2. It is important to mention that the caregivers themselves will not have access to the data that has been decrypted during this transcription process, since the information will be internally encrypted and sent right away.

The reason for the new policy evaluation on the decrypted data is that it may be possible that $C2$ has different access rights from $C1$, which means that $C2$ may be allowed to see the information that $C1$ has received in encrypted form.

3.4 Key Management

This section describes how the keys and their shares are managed, including information about the key share distribution and gathering as well as the key lifecycle.

3.4.1 Key Distribution

Whenever the document sections are encrypted, the key will be divided into n shares according to Shamir secret-sharing scheme (see Section 4.4). These n shares will be distributed among the caregivers on the peer-to-peer network. There are no rules on who would receive a key share, but it seems to be most logical to distribute the key shares as followed. One share will stay with the caregiver who is initiating the information exchange and the other shares are distributed to caregivers, who are collaborating caregivers of the originating caregiver and who are preferably involved with the patient. In case the originating caregiver does not have enough collaborating caregivers, the shares will be distributed among other caregivers on the network. Figure 3.4 shows an example where an encrypted document is sent from Family Doctor (D) to Specialist (S1) using a (3,4)-threshold sharing scheme, which mean that 3 out of 4 shares are needed to reconstruct the key. D, S1 as well as the Specialist (S2) and the Walk-in clinic (W) are receiving a key share.

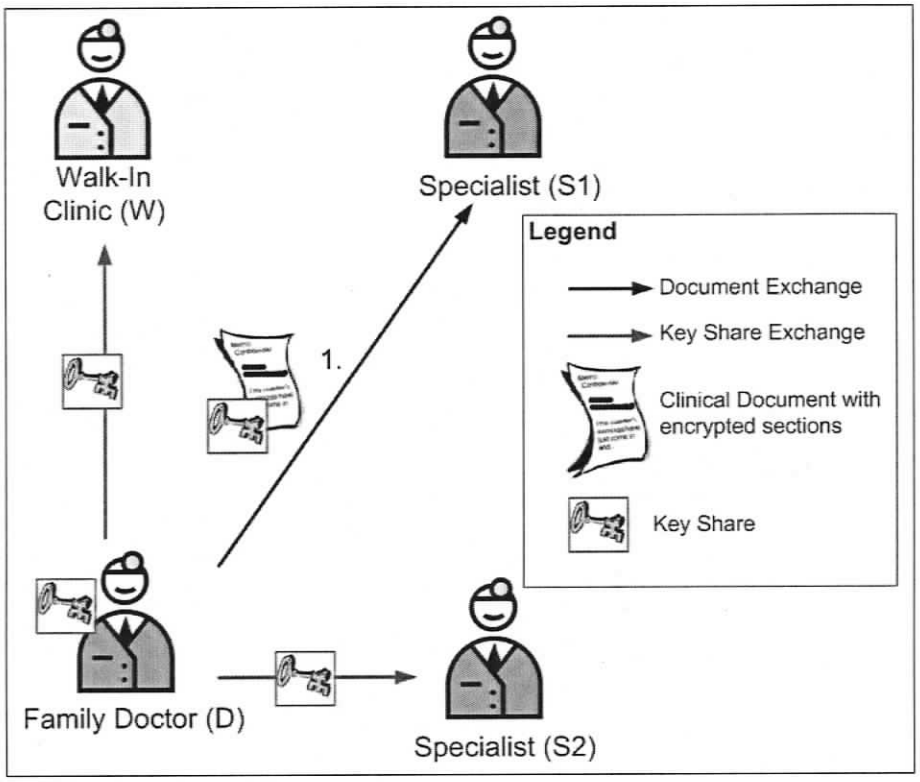

Figure 3.4. *Key sharing mechanism*

3.4.2 Key Share Gathering

Whenever caregivers see the need to access the encrypted sections, they would verify whether given the new reason they are allowed to access the encrypted data. This policy evaluation is done by the CDAShip. If the access to the encrypted section under the new reason is still not granted, the caregiver will get an option to choose whether the section has to be accessed in any case or whether it is not as important that the privacy has to be violated. In the first case the caregiver will send a key share request message to all caregivers that own a share of this key. This request message includes the reason why the information should be accessed as well as the result of the policy evaluation. The result of the policy evaluation is needed for the audit trail and to determine whether the access would have been granted according to the policies. It is assumed that the caregivers are given access to the information in any case. However, the patient will be notified, e.g., via

email that the information has been accessed, by whom and for which reason. If the patient disagrees that the given reason is enough for a privacy violation then the patient is able to take action and hold the caregiver to account.

If we consider that the key shares have been exchanged as shown in Figure 3.4 and S1 wants to access the encrypted information, then S1 requests the shares from D, S2 and W, a shown in Figure 3.5. The request includes the document ID as well as the reason why the data has to be accessed.

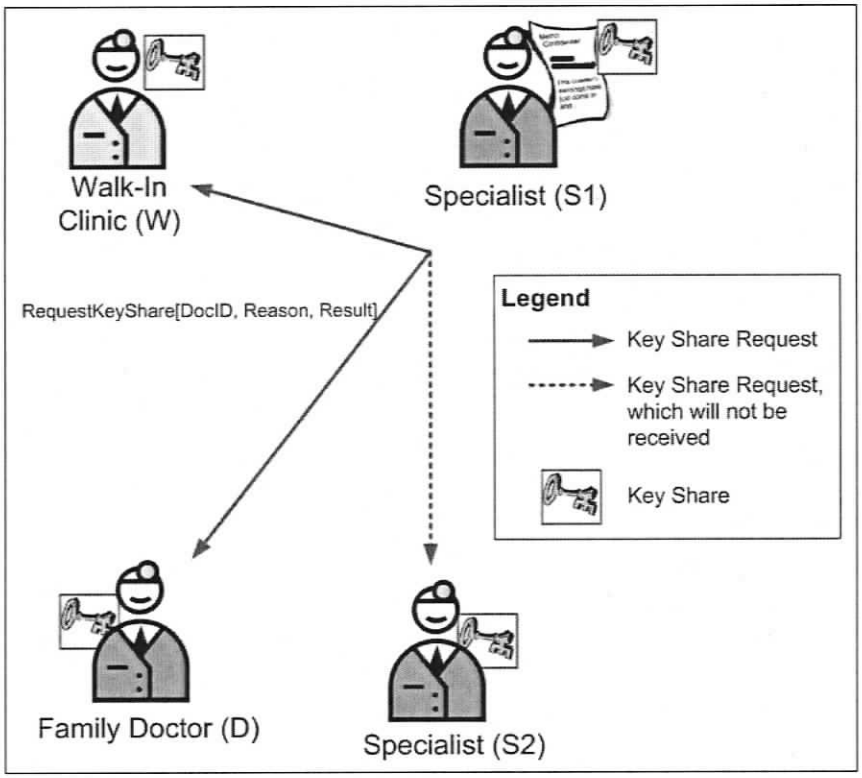

Figure 3.5. *Key gathering notification*

In this example, it is further assumed that D and W received the request, but S2's information system is currently not available and therefore cannot reply to the request. D and W will inform the patient about the request and indicate which information will be accessed by S1 and for which reason. Also they will record the request in a database before sending their key shares to S1. Having received the key shares from D and W, S1 now has

three shares and is able to reconstruct the key.

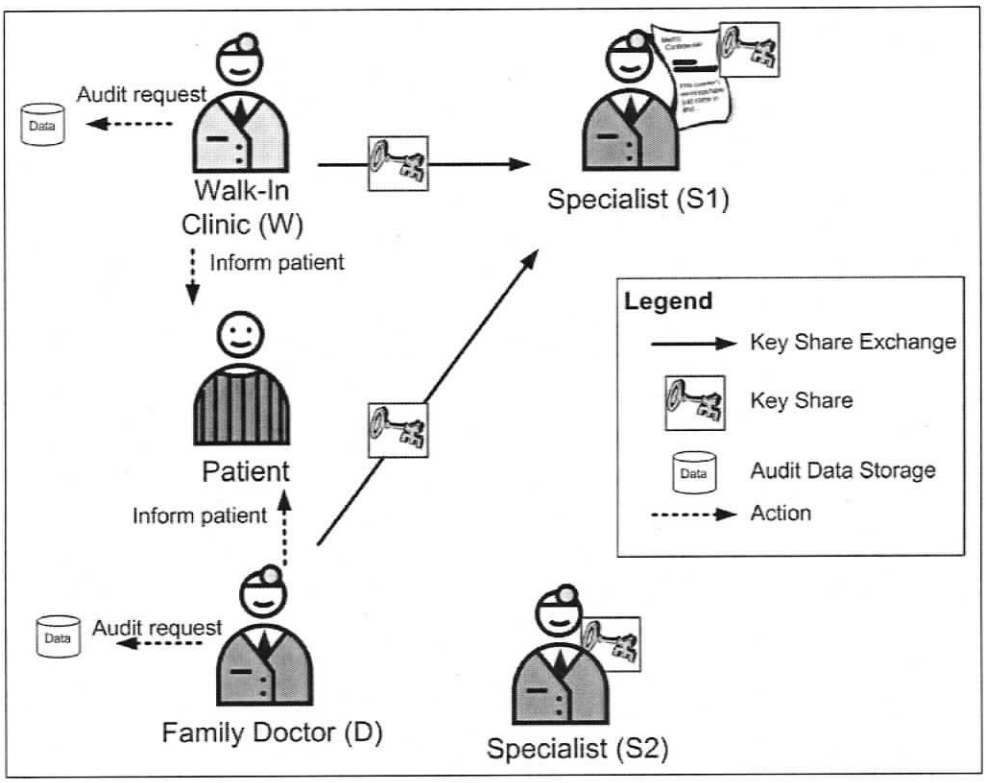

Figure 3.6. *Key Share Exchange*

There will be no further notification about whether S1 has received enough shares to reconstruct the secret key and actually get to the protected data, since there would be no proof about whether this notification would be true or not. Since S1 could pretend to not have gotten enough shares, but in reality has received at least t shares and therewith could access the sensitive data. That is why it is always assumed that S1 has accessed the confidential data after requesting key shares.

3.4.3 Key Lifecycle

Due to the fact that an electronic information exchange on a peer-to-peer network is fast and inexpensive it is assumed that many documents will be exchanged among the caregivers.

However, the more information is exchanged the more key shares a caregiver has to store. To prevent that a caregiver stores needless information each key has a certain lifetime. Once this lifetime is expired the CDAShip may automatically delete the key shares, since it is assumed that the encrypted information is no longer needed. In case the encrypted information is needed after the key has expired, it is still possible to request the information from the caregiver that holds the original information.

Chapter 4

Foundation

4.1 Health Level 7

Health Level 7 (HL7) [1] is one of the world's biggest health care standards developers. Its name "Health Level 7" refers to the application-to-application interface defined in the top layer (Level 7) of the Open Systems Interconnection (OSI) layer protocol for the health environment. A small committee comprised of healthcare providers, software vendors and consultants founded the HL7 organization in March 1987 with the goal to link different parts of a hospital such as patient administration, laboratory and billing. Since its creation, the HL7 organization has grown from a user-based consensus organization to an international organization with affiliate groups in Australia, Canada, Finland, Germany, India, The Netherlands, New Zealand, South Africa and the United Kingdom. HL7's mission is *to provide standards for the exchange, management and integration of data that support clinical patient care and the management, delivery and evaluation of healthcare services. Specifically, to create flexible, cost effective approaches, standards, guidelines, methodologies, and related services for interoperability between healthcare information systems* [1]. The interoperability between medical information systems is one of the major issues when integrating various information systems among medical organizations. Therefore, organizations such as HL7 that create standards for data and processes between healthcare providers and vendor systems are needed.

4.1.1 Clinical Document Architecture

The HL7 Clinical Document Architecture (CDA) [12] is a combination of standards for the representation of clinical documents (such as discharge summaries and progress notes). It is a document markup standard, which specifies the structure and semantics for clinical documents. These CDA documents can be exchanged within an HL7 message, but they also can exist independently outside a transferring message. Furthermore, they can include text, images, sounds and other multimedia content. Additionally, CDA documents are persistent, stewarded, human readable and intended to be legally authenticated. Another feature is that they are application- and platform-independent and therewith they can be viewed and modified by various tools.

4.1.1.1 CDA Document Structure

As member of the HL7 version 3 family of standards, the CDA document semantic is derived from the HL7 Reference Information Model (RIM) and implemented in the Extensible Markup Language (XML) [41]. Currently three types of CDA documents are distinguished [12]. These types are described below [13].

CDA Level One documents consist of a document header, which is specified in detail, and a document body that is one narrative clinical block. The Level One specification has only one schema definition for all different types of clinical documents. Its intention is to minimize the adoption barriers of the standard.

CDA Level Two is a set of templates, which enables the ability to constrain the set of allowable sections based on document type.

CDA Level Three adds additional markup to Level Two in order to define observations and services within the document body, with a granularity as rich as is possible with the RIM.

The actual clinical content of a CDA document is independent from the CDA level, meaning that is it possible to express the same content in a CDA Level One document

and in a Level Three document. The three levels differ in the degree to which a machine can process the document and to which degree constraints can be added to the document's content.

4.1.1.2 CDA Document Components

A CDA document contains two parts, a document header and a document body. The document header includes information on creation of the document and is the same for each document type whereas the actual document content is included in the document body.

The intention of the document header is to facilitate

- the information exchange across and within organizations,
- the compilation of individual's clinical record, and
- clinical document management.

Furthermore, the header includes four components:

Document information is used to identify the document, define the confidentiality status as well as the relationship to other documents.

Encounter data specifies the settings in which documented encounter occurred.

Service actors contains information about document originators, authenticators, receivers and other health care providers who participate in the services being documented.

Service targets includes all significant participants such as patient and his/her family members.

The CDA document body contains the actual clinical data and can be structured or unstructured depending on the CDA level (see Section 4.1.1.1). Structured bodies encoded in XML consist of CDA structures such as sections, paragraphs, lists and tables, which can be nested.

4.2 Policies

Matt Bishop defines security policy as a statement that partitions the states of the system into a set of authorized, or secure, states and a set of unauthorized, or nonsecure, states [8]. The policies address individual consents, organizational procedures and legislative regulations. Furthermore, policy-based management promises to be an adaptable solution for distributed systems that are able to dynamically change their behaviour whenever the policies change [19].

In health care three types of policies can be distinguished.

Patient policy: The individual patients determine who is allowed to access their information.

Organizational policy: The organizations determine which information is allowed to be shared with collaborating organizations. Organizational policy will also include governmental and legislative regulations.

Document type policy: Different document types (e.g., blood test) have different criteria for information exchange, due to the different content sensitivity of documents.

In order to ensure that just authorized information will be exchanged, it is necessary to combine all policies that affect the document and determine whether the information exchange is permitted or denied according to the policies. A combination of different policies often results in conflicts. E. Lupu and M. Sloman describe in [27] how to resolve policy conflicts using precedence. One suggested solution is that negative policies override positive policies or vice versa. An example for a negative policy could be "Do not exchange information from patient X", whereas a positive policy could be "The exchange of the blood type of patient X is permitted in emergency cases." If a negative policy overrides a positive one then in this case no information is exchanged even in emergencies. Another solution is that the priority is set dependent on how specific the policy is, which means that policies that apply to specific sections of a document override policies, which apply to the whole document. Still not all conflicts would be resolved, since it is possible that there are policies

that apply to the same section.

4.2.1 Policy Description Languages

In order to express a policy in a machine-interpretable and unambiguous form it is necessary to use a policy language that is based on a formal model for authorization management and access control for personal information [4]. Standard policy description languages enable cross-organizational policy enforcement for protected data that is shared among collaborating organizations. This section introduces different policy description languages.

4.2.1.1 P3P

The World Wide Web Consortium (W3C) developed the Platform for Privacy Preferences Project (P3P) [35] in order to fulfill the need for a simple, automated mechanism for internet users to gain more control over the use of personal information on Web sites they visit.

Using P3P an organization can state their privacy practices using a standard XML-based format. A user's P3P enabled browser is able to "read" the privacy policy, compare it to the user's own set of privacy preferences and take appropriated actions.

Even if P3P provides a technical mechanism for ensuring that users can be informed about privacy policies prior to the release of personal information, it does not ensure that sites act according to their policies.

However, P3P focuses on privacy promises rather than on privacy realization and it is not used for information exchange among collaboration organization. Therefore, P3P is not suitable for the approach described in this thesis.

4.2.1.2 EPAL

The Enterprise Privacy Authorization Language (EPAL) [16] was developed by IBM and submitted to the W3C in November 2003. It is an XML-based mark up language created to enable organizations to translate their privacy policies into IT control statements and

enforce policies that may be declared and communicated using P3P.

An EPAL policy comprises of two top components, the EPAL-vocabulary and the EPAL-policy. The EPAL-vocabulary defines vocabulary elements used to specify policies. Organizations that employ EPAL as a policy enforcing mechanism declare a specific vocabulary based on their needs. The EPAL-policies then define the policy rules based on an organization's EPAL-vocabulary. In order to share information and their policies, collaborating organizations would have to have the same set of vocabulary elements to facilitate privacy enforcement, which makes it unsuitable for distributed computing.

4.2.1.3 XACML

The Organization for the Advancement of Structured Information Standards (OASIS) approved the eXtensible Access Control Markup Language (XACML) as a standard access control policy language in February, 2003 [43]. It provides a flexible mechanism for expressing and enforcing access control policies through a single language.

XACML defines syntax for a policy language and semantics for processing the Policies. There is also a request and response format to query the policy system, and semantics for determining applicability of policies to requests [26].

Basically, XACML is a core set of XML schemas along with a corresponding namespace. This set of schemas expresses authorization rules that are combined to form authorization policies. XACML policies can then be compared to other XML documents in order to determine the authorizations that should be given to various parties.

A typical scenario for using XACML policies could be that someone wants to access information and sends it to a third party. The sender would send a request to whatever actually protects this resource (e.g. web server), which is called a Policy Enforcement Point (PEP). The PEP will create a request based on the requester's attributes, the resource in question, the action, and other information pertaining to the original request. The PEP will then send this request to a Policy Decision Point (PDP) that will use the request and the policies that apply to the request, to determine whether access should be granted or not.

The result will be returned to the PEP, which then can allow or deny access to the requester.

XACML is a standard policy language used to describe general access control policies. Furthermore, the fact that XACML is defined as XML schema and the CDA are encoded in XML, makes XACML the most appropriated policy languages for securing clinical information exchange.

4.3 Confidentiality Mechanisms

This section introduces mechanisms used to ensure confidential information exchange. Section 4.3.1 describes cryptography mechanisms that are mandatory to ensure that just authorized people are able to access the information and that the information has not been modified during the transfer. Whereas Section 4.3.2 explains an authentication mechanism.

4.3.1 Cryptography

Cryptography is essential to security in distributed systems, since it provides the basis for [37]

Authentication: Proving someone's identity.

Privacy / Confidentiality: Ensuring that no one can read the message except the intended receiver.

Integrity: Assuring the receiver that the received message has not been altered in any way from the original.

Non-repudiation: The inability to deny the integrity and authenticity of a message.

Therewith cryptography is necessary when handling sensitive information within a non-trusted medium; this includes amongst others the medium where the information is stored as well as the medium used to transfer, e.g. the Internet. Basically cryptography is the art of encrypting information before it is sent to or stored on the medium and decrypting

it after the information has been retrieved from the medium. Encryption and decryption are realized by using cryptographic mechanisms. This section describes three types of cryptography as well as the message exchange over secure channels.

4.3.1.1 Types of Encryption

In general there are three types of cryptographic mechanisms, which are typically used to accomplish authentication, privacy/confidentiality, integrity, and non-repudiation

- secret key (or symmetric) encryption,
- public-key (or asymmetric) encryption,
- hash functions.

Secret key (or symmetric) encryption

Secret key encryption [3] is characterized by using one single key to perform both encryption and decryption of data. The sender encrypts its message using the secret key and the recipient who receives the encrypted message uses the same secret key to decrypt the message. Figure 4.1 visualizes this scenario. Due to the fact that the symmetric key encryption algorithms are public knowledge, the security is determined by the level of the key protection (i.e., ensuring that the key is known only to the parties involved in information exchange).

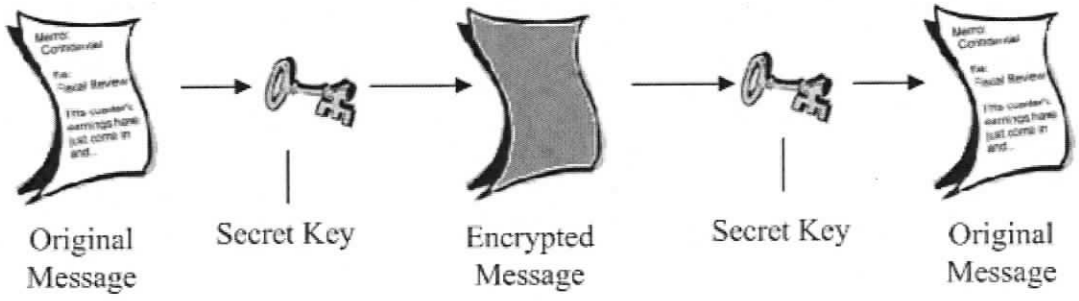

Figure 4.1. *Secret key encryption process*

Public-key (or asymmetric) encryption

The Public Key Encryption system [3], first proposed in 1976, involves two asymmetric cryptographic keys: one is a public key, which is maintained in a public key repository open to anyone who asks for it, and the other one is a private key that the owner maintains and which is the owner's unique identity. These two keys are separate and distinct, but still mathematically related. The mechanism depends on the fact that anyone can encrypt a message using a public key, but that the matching private key is needed to decrypt the message. The following is a very brief example.

Consider that Bob wants to send a message securely and secretly to Alice. Therefore, he encrypts his message using Alice's public key and sends the encrypted message to Alice. Alice is the only one who can decrypt this message, since she is the only on who knows her private key and therewith Bob can be sure that nobody else can read his message.

Figure 4.2 illustrates the public key encryption process.

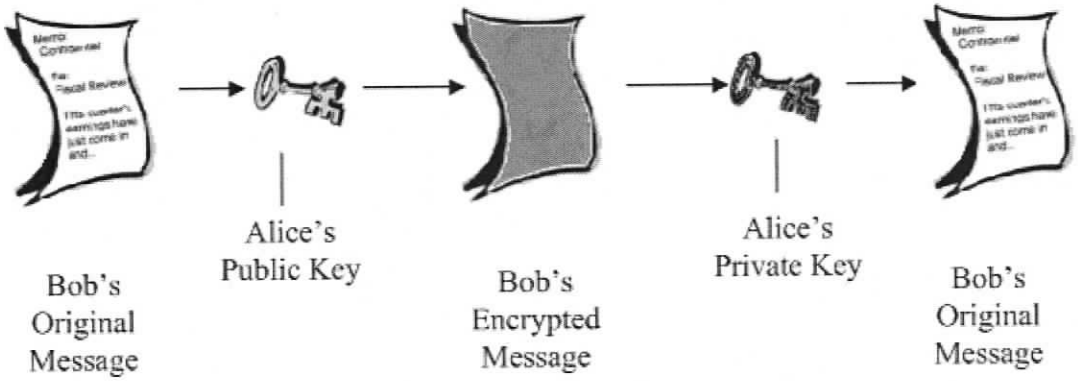

Figure 4.2. *Public key encryption process*

Hash function

Cryptographic hash functions map a large data object of variable size to a small data object of a fixed size. Furthermore, they are one-way functions, which

means that it is infeasible to find any input which maps to any pre-specified output. In addition to that, they are also collision-less, therefore it is nearly impossible to find any two distinct inputs which map to the same output. If we assume a cryptographic hash function computes a hash value of 128 bits of a message, then the probability of finding a message corresponding to a given hash is 2^{-128} and the probability of finding two messages with the same hash is 2^{-4} [8]. Since these probabilities are very small it is common to use cryptographic hash functions as a "fingerprint" of a message in order to provide integrity.

4.3.1.2 Secure Channel

In order to provide a secure communication within a distributed system, each communication message has to be transmitted via a secure channel. In Section 4.3.1.1 different types of encryption methods were introduced. This section describes how to use these types to create a secure communication channel.

The secret key encryption (see Section 4.3.1.1) has the disadvantage that the used keys have to be known to both parties and therewith they also have to be exchanged. In contrast to this, using the public key encryption (see Section 4.3.1.1) does not require this kind of key exchange, which decreases the chances that a key can be stolen or used by somebody else. However, a disadvantage of using public key cryptography is the speed of the encryption and decryption process. Secret key encryption methods are significantly faster than public key encryption methods. Therefore, it is usual in network communication to encrypt the message, which is exchanged, with a secret key and then to encrypt the secret key using the public key encryption. In this way it is possible to get the best out of both encryption mechanisms, the fast encryption using secret keys and the enhanced security from the public key encryption.

4.3.2 Authentication

Matt Bishop defines authentication, in his book Computer Security [8], as a binding of an identity to a subject. In other words authentication means proving that a subject (e.g. message sender), is what it claims to be.

This section describes the digital signature mechanism, which is used to authenticate the origin of a received message or document.

An overview of other authentication protocols used in distributed systems can be found in [25].

4.3.2.1 Digital Signature

A digital signature is analogous to a handwritten signature since just a single entity can sign data and multiple entities are able to read it. The digital signature itself is a sequence of bits, which is appended to the end of the message. Generating those bits using cryptographic algorithms gives the digital signature bit sequence a meaning in contrast to just any bit sequence.

Digital Signatures are based on the public key approach (see Section 4.3.1.1), only the circumstances are a bit different. While using the public key encryption Bob was ensuring that no one other than Alice could decrypt and therewith read his message, using digital signatures Bob wants to send a message to Alice and he wants to be able to prove that it really came from him. But, he does not care whether anybody else reads his message. In order to create a digital signature, Bob will encrypt his message with his own private key that nobody else knows, since it has been kept secret since generation time. He will attach the encrypted message to the original one and send these two messages together to Alice. Alice, and any other recipient, is now able to decrypt the encrypted message signature using Bob's public key. Furthermore, she can compare the now decrypted message with the original message that was sent. In the case both messages are identical it is proven that the message is really from Bob, because Bob was the only one who could have signed the

message using his private key. If both messages are not identically it is proven that the message was not signed by Bob or that it has been compromised during the transfer. Figure 4.3 visualizes the digital signature scenario.

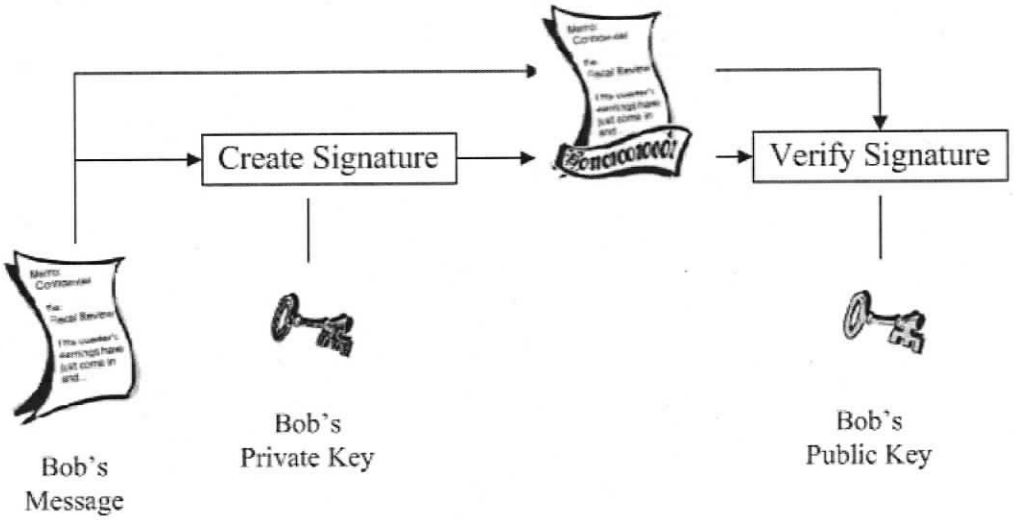

Figure 4.3. *Generic digital signature process*

Applying a cryptographic hash function (see Section 4.3.1.1) to the message before signing it, can improve the digital signature mechanism in two ways. First of all hashing is generally much faster than public-key encryption, so encrypting a hashed message will save the sender and receiver time. Second it will save space since even an encrypted message digest of a message is much shorter than the encrypted version of the entire message. The receiver will apply the same cryptographic hash function to the message and compares the resulting message digest to prove the authenticity and integrity of the message.

Digital signatures are accepted as legal evidence using the same general guidelines as hand-written signatures. In addition to that, the signature verification is more accurate compared to a hand-written signature, because it uses the public key verification. But still since the security of digital signatures relies on the management of the signature keys, it is important to check the validity of the digital signature in case the keys are compromised [23].

4.4 Secret-sharing

Any secure system, which uses cryptography, has to ensure that the keys used to secure the data are protected as well. Possibly by using other keys if the original keys are stored in a physically insecure location. However, the keys used for protecting the keys have to be protected themselves, so regardless how many keys a system uses, it can not be avoided that there exist one or more keys in the system, which are only protected, because they are stored in a physically secure manner. Since a key is so important, it would not be good if the key was revealed to an adversary. It would be equally bad, if the key would be lost or is not retrievable anymore. This means that a security mechanism has to ensure on the one hand that keys are not revealed to anyone and stored only in a single, but very secure location and on the other hand that the keys are always available, which seems to imply that they should be stored in as many different locations as possible. The secret-sharing technique addresses both of these issues at the same time.

4.4.1 Concept and Definitions

Secret-sharing schemes are multi-party protocols used for key establishment. Initially it was used to create backup copies to prevent loss of keys. However, the more keys exist the bigger is the risk of security exposure. Secret-sharing schemes are methods for distributing a secret, such as a key, amongst a group of participants, each of which is allocated a share of the secret. Each individual share is of no use on its own, the only way to reconstruct the secret is to combine the shares.

A *perfect* secret-sharing scheme is a method of sharing a secret S among a set of users in such a way that the following two properties are satisfied:

1. If an authorized subset of users pools their shares together, they can recover the secret S .
2. If an unauthorized subset of users pools their shares together, they are unable to determine the secret S neither are they able to gain any information on S .

The *information rate* for a specific user in a secret-sharing scheme is defined as the bit size ratio $\frac{size\ of\ the\ shared\ secret}{size\ of\ the\ user's\ share}$. Whereas the information rate of the sharing scheme is the minimum information rate of all the users in the scheme. A scheme with an information rate 1 is said to be ideal, since the size of a share is equivalent of the size of the secret.

4.4.2 Simple Shared Control Schemes

It is possible to use secret-sharing scheme as a shared control scheme [29] if the shares of two or more users are needed to enable a critical action such as the opening of a bank vault or the firing of a missile. In shared control schemes the secret can only be computed if all shareholders get together and pool their shares.

Further, shared control schemes are based on modular or exclusive-Or addition. It is distinguished between dual control schemes, which requires 2 out of 2 users to retrieve the secret, and unanimous consent control, which needs t out of t users to reconstruct the secret.

4.4.2.1 Dual control by modular addition

The secret division and share distribution mechanism consists of the following steps

1. Choose secret S and a random integer with $0 \leq S \leq m-1$.
2. Select a random number S_1 with $1 \leq S_1 \leq m-1$
3. Distribute S_1 and $(S - S_1) \text{mod } m$ to users P_1 and P_2 respectively.

It is further assumed that the users P_1 and P_2 are trusted not to conspire together to ensure that neither user has any information about S and that both separately enter their share values into the device, which sums them modulo m and recovers S.

4.4.2.2 Unanimous consent control by modular additions

It is possible to generalize the dual control scheme to the case that t user shares are needed in order to recover the secret S. The secret division and share distribution mechanism is as

follows

1. Choose secret S and a random integer, m, with $0 \leq S \leq m-1$.
2. Select $t-1$ random independent numbers S_i with $0 \leq S_i \leq m-1$,$1 \leq i \leq t-1$
3. Distribute S_i to user P_i, $1 \leq i \leq t-1$ and $S_t = S - \sum_{i=1}^{t-1} S_i \text{mod } m$ to user P_t

Again it is assumed that all t users separately enter their share values into the device which sums them modulo m and reconstructs the secret since $S = S - \sum_{i=1}^{t} S_i \text{mod } m$.

It is not possible for $t-1$ users with set $P \setminus \{P\}_i$ $(1 \leq i \leq t-1)$ to recover the secret S, since they only own the shares $S_1, ..., S_{i-1}, S_{i+1}, ..., S_{t-1}$ and $S = S - \sum_{i=1}^{t-1} S_i$. The device would only be able to compute the value $S - Si$, since the random number S_i is missing. Therewith, S cannot be computed.

4.4.3 (t,n)-Threshold Scheme

A (t,n)-threshold scheme ($t \leq n$) [29] is a cryptographic scheme in which a datum is divided into n shares and any t of these n shares are sufficient to determine the original datum. A threshold scheme is perfect if the knowledge of t-1 or less shares provides no knowledge about the secret. In the sense that an opponent that has t-1 shares knows as much, to wit nothing, about the secret as an opponent with no shares at all.

An adversary would have to get hold of at least t shares in order to reconstruct the key. Also, as long as not more than n-t-1 shares are lost, it is possible to reconstruct the key. Therefore, (t,n)-threshold schemes are an ideal solution for key management, since they are at the same time robust against lost of information as well as information leakage.

4.4.4 Generalized Secret-Sharing

In general any secret-sharing scheme can be defined by a set P of all users and A (the *access structures*) a set of subsets, so called *authorized subsets*. *Authorized subsets* are the subsets of users in P, which are able to compute the secret S.

(t,n)-threshold schemes are special cases of generalized secret-sharing schemes, in which the access structure consists of exactly all t-subsets of users. Their access structure is defined as followed

$$A = \{B \subseteq P :| \ B \ | \geq t\}$$

Assuming that $B \in A$ and $B \subseteq C \subseteq P$. A is called *monotone* if for all C the secrect S can be reconstructed by ignoring the shares of the participants in $C \backslash B$, because B is an *authorized subset* and can already determine the secret.

4.4.5 Shamir's Scheme

In his 1979 paper "How to Share a Secret" [38], Shamir introduced his (t,n)-threshold scheme, which is based on Lagrange interpolating polynomials.

The Lagrange interpolating polynomial is the polynomial $f(x)$ with a degree less than n that passes through the n distinct points (x_1, y_1),(x_2, y_2), ..., (x_n, y_n). Furthermore, $f(x)$ is given by

$$f(x) = \sum_{i=1}^{n} y_i P_i(x) \text{ with } P_i(x) = \prod_{k=1, k \neq i}^{n} \frac{x - x_k}{x_i - x_k}$$

written explicitly,

$$P(x) = \frac{(x-x_2)(x-x_3)...(x-x_n)}{(x_1-x_2)(x_1-x_3)...(x_1-x_n)} + \frac{(x-x_1)(x-x_3)...(x-x_n)}{(x_2-x_1)(x_2-x_3)...(x_2-x_n)} + ... + \frac{(x-x_1)(x-x_2)...(x-x_{n-1})}{(x_n-x_1)(x_n-x_2)...(x_n-x_{n-1})}$$

The Shamir (t,n)-threshold scheme is defined for a secret $s \in \mathbb{Z}/p\mathbb{Z}$ with p prime $(p > max(s, n))$ and $a_0 = s$. Further $a_1, ..., a_{t-1}$ are chosen randomly in $\mathbb{Z}/p\mathbb{Z}$ from the trusted party that created the secret and computes $f(i)$, with

$$f(i) = \sum_{k=0}^{t-1} y_{t-1} a_k x^k \ \forall \ 1 \leq i \leq n$$

The shares $(i, f(i))$ are then distributed to the n distinct parties. Due to the fact that the secret is the constant term $s = a_0 = f(0)$ it can be recovered from any t shares $(i, f(i))$ using

$$s = \sum_{i=1}^{t} c_i f(i), \text{ with } c_i = \prod_{1 \leq j \leq t, j \neq i} \frac{i}{j-i}.$$

Since c_i are non-secret constants, which may be pre-computed, it is possible for everybody who owns t shares $f(i)$ to reconstruct the secret s.

Additionally the Shamir (t,n)-threshold scheme has the following properties.

Perfect: with knowledge of t-1 or less shares, all values of S remain equally probable.

Ideal: Size of a share is the size of the secret.

No unproven assumptions: Does not rely on presumably hard problems.

Extendable for new user: New shares can be computed and distributed without affecting existing shares.

Varying levels of control possible: Multiple shares can be given to a single individual

4.4.6 Blakley's Scheme

In the same year Shamir [38] devised his threshold scheme, Blakley [9] invented his own (t,n) threshold scheme which is similar in nature to Shamir's scheme even though both discoveries were made independently. Blakley's secret-sharing scheme is geometric in nature. The secret is a point in a t-dimensional space. n shares are constructed with each share defining a hyperplane in this space. By finding the intersection of any m of these planes, the secret (or point of intersection) can be obtained. This scheme is not perfect, as the person with a share of the secret knows that the secret is a point on his hyperplane. However, this scheme can be modified to achieve perfect security.

4.4.7 Verifiable Secret-Sharing

In all presented secret-sharing schemes it was assumed that a trusted dealer would create the shares and distribute them. However, a misbehaving dealer can distribute inconsistent shares to the participants, from which they will not be able to reconstruct the secret. In order to prevent such malicious behavior of the dealer, it is necessary to implement a protocol which allows to verify a consistent dealing by the users.

It is said that a set of n shares $S_1, S_2, ..., S_n$ is t consistent if any subset of t out of n shares defines the same secret S [6].

Problematic with Verifiable Secret-Sharing is that the users have to be convinced that their shares (collectively) are t-consistent. In Shamir's scheme [38], the distributed shares $S_1, S_2, ..., S_n$ are t - consistent if and only if the interpolation of the points $(1, S_1), (2, S_2), ..., (n, S_n)$ yields a polynomial of degree at most $k - 1$. This means that all n shares are needed to verify the secret, which stands in contrast with a (t,n)-threshold scheme and the fact that only t shares are needed to construct the secret.

There are two types of Verifiable Secret-Sharing protocols, the interactive and Non-interactive proofs, which both allow the verification of the secret shares without them being revealed. Furthermore, both protocols are based on the following property

- If the sum of two polynomials is of degree at most $t - 1$, then either both are of degree at most $t - 1$ or both are of degree greater than $t - 1$.

4.4.7.1 Interactive Proof

This section describes two different interactive proofs for verifiable secrets sharing schemes as described by Benaloh [6].

Trusted Users and Distributed Dealer

This protocol assumes that the users are trusted and do not cheat. The protocol is as follows

1. The Dealer uses Shamir's secret-sharing scheme for a secret S in order to create $f(x)$, in which $d(0) = S$, and distributes the shares $S_i = f(i)$ to the participant P_i $(1 \leq i \leq n)$.
2. The Dealer chooses many (e.g. 100) random polynomial $P_1(x), ..., P_{100}(x)$ of degree $t-1$. Further the Dealer commits himself to the polynomials by distributing to every ith shareholder 100 shares, one of each polynomial $P_1(i), ..., P_{100}(i)$ $(1 \leq i \leq n)$

3. The shareholders choose 50 random indices from [1..100], e.g. $\{j_1, ..j_{50}\}$, and ask the Dealer to reveal the polynomials $P_{j1}(x), ..., P_{j50}(x)$.
4. The Dealer reveals the polynomials in the chosen subset $P_{j1}(x), ..., P_{j50}(x)$. It is assumed that all the shareholders receive the same polynomials.
5. The shareholders verify that the distributed shares of the revealed polynomials define polynomials of degree $t-1$.

So far, the dealer has proven (with high probability) that all the shares of all the polynomials $P_1(x), ..., P_{100}(x)$ define polynomials of degree $t-1$.

1. The Dealer reveals $f(x)+P_{j51}(x), ..., f(x)+P_{j100}(x)$, where $\{j_{51}, ..j_{100}\}$ are the remaining indices.
2. The shareholders verify that $f(x)+P_{j51}(x), ..., f(x)+P_{j100}(x)$ are of degree $t-1$ and match to their own shares, e.g. the i'th shareholder verifies, using the homomorphic property, that $f(i) + P_t(i)$ (simply by summing his corresponding shares) equals to $f(x) + P_t(x)$ in the i'th coordinate (by substituting the i'th coordinate in the revealed polynomials' sum) for all $t \in \{j_{51}, ..j_{100}\}$.

Untrusted Shareholders, Untrusted Dealer

This protocol makes no assumption about the trustworthiness of neither Dealer nor shareholders. The general idea is that the Dealer will commit by encryption and instead of distributing the shares (of the 100 random polynomials) to every shareholder, the Dealer will encrypt all the shares and then exchange them all. Furthermore, there are two additional requirements for this protocol

1. The encryption algorithm should have the homomorphic property $E(x \times y) = E(x) \times E(y)$ such as the Diffie-Hellman encryption.
2. The Dealer should use a secure broadcast while publishing the encrypted shares.

The protocol is as follows

1. The Dealer uses Shamir's secret-sharing scheme for a secret S in order to creates $f(x)$, in which $d(0) = S$, and distributes the encrypted shares $S_i = f(i)$ to the participant P_i $(1 \leq i \leq n)$.
2. The Dealer chooses many (e.g. 100) random polynomial $P_1(x), ..., P_{100}(x)$ of degree $t - 1$. Further the Dealer commits himself to the polynomials by publishing the 100*n encrypted shares $E(P_1(1)), ..., E(P_{100}(1)), ...,$ $E(P_1(n)), ..., E(P_{100}(n))$
3. The shareholders choose 50 random indices from [1..100], e.g. $\{j_1, ..j_{50}\}$, and ask the Dealer to reveal the polynomials $P_{j1}(x), ..., P_{j50}(x)$.
4. The Dealer reveals the polynomials in the chosen subset.
5. The shareholders verify that $g^{P_w(i)} = E(P_w(i))$, for all $w \in \{j_1, ..j_{50}\}$ and for all $i \in [1..100]$
6. The Dealer reveals $f(x)+P_{j51}(x), ..., f(x)+P_{j100}(x)$, where $\{j_{51}, ..j_{100}\}$ are the remaining indices.
7. Each i'th ($i \in [1..100]$) shareholder uses the homomorphic property to verify that $g^{f(x)+P_w(i)}$ is equals to $E(f(i)) * E(Pt(i))$ (for all $t \in \{j_{51}, ..j_{100}\}$)

Disadvantages

The interactive proof has two disadvantages.

First of all it useful for the participants of this protocol, and only at the moment it is held. For users that are currently not online or do not participate in the random selections this proof has no meaning at all.

Second, due to the first disadvantage this proof can not be used as a legal proof, since it is not valid to any third party.

4.4.7.2 Non Interactive Proof

In [17] Feldman describes a non-interactive protocol to verify the secret shares. In this protocol only the dealer is allowed to exchange messages. The users neither talk to the dealer nor to each other during the verification process.

In addition to the share the dealer sends extra information to each user when exchanging the shares. Further each user verifies that his/her share is consistent with the extra information.

In addition, this protocol requires that the used encryption mechanism has the homomorphic property, with respect to both addition and multiplication $E(x \times y) = E(x) \times E(y)$ and $E(x + y) = E(x) + E(y)$ such as the Diffie-Hellman encryption algorithm.

The protocol is as followed

1. The Dealer uses Shamir's secret-sharing scheme for a secret S in order to creates $f(x) = a_0 + a_1 x + ... + a_{t-1} x^{t-1}$, in which $d(0) = S$, and distributes the shares $S_i = f(i)$ to the participant P_i $(1 \leq i \leq n)$. In addition, the dealer publishes the encrypted shares $E(a_0), ..., E(a_{t-1})$ of all t coefficients of $f(x)$ as well as the generator g and the field $\mathbb{Z}_p$ of the encryption mechanism.
2. Each user $P_i (1 \leq i \leq n)$ verifies his/her own share by checking the following equation

$$E(f(i)) = E(a_0 + (E(a_1 \times E(i)) + ... + (E(a_{t-1}) \times E(i^{t-1}))$$

   This is done based on the homomorphic and associative properties for both addition and multiplication.
3. If this equation holds, user P_i broadcasts a message saying that he/she accepts his/her share as proper. If all the shareholders find their shares correct then the dealing phase is completed successfully. If a user P_j finds the above equation incorrect then user P_j publishes an accusation against the dealer. The honest shareholders can decide whether it is the dealer or the accuser that is faulty.

4.4.8 Proactive Secret-Sharing

Proactive schemes are seeking to make the secret-sharing mechanism more secure by regularly updating the secret shares. The main idea behind it is that is is assumed that attackers need some time and effort to steal or destroy all needed shares in order to retrieve the secret and that before they can obtain the threshold number, all shares will have been actively changed or overwritten in a way that any information about the individual shares becomes obsolete and useless once the shares are renewed. It is important to mention that the periodic renewal of the shares is done without changing the secret. Another aspect of the proactive secret-sharing scheme is that is tries to avoid the gradual destruction of the information by corruption of the shares. This is done by periodically recovering lost or corrupted shares without compromising the secret of any shares [20].

The Proactive Secret-Sharing has the following requirements:

- An adversary is not able to obtain more than t-1 shares in any time period (where $t - 1 < \frac{n}{2}$, which guarantees the existence of t honest shareholders at any given time). This time period should be synchronized with the share-renewal protocol.
- Authenticated broadcast channel.
- Authenticated and secret communication channels between each two participants.
- Synchronization: the shareholders can access a common global clock so that the protocol can be applied in a certain time period.
- Shares can be erased: every honest shareholder can erase its shares in a manner that no attacker can gain access to erased data.

4.4.8.1 Basic Share Renewal Protocol

The basic Share Renewal Protocol renews the secret shares without involving the original Dealer, since it is possible that the Dealer does not exists anymore. Further, all shareholders have to agree on a new polynomial with the same secret S without revealing the secret, the old polynomial or the new polynomial. Finally, each shareholder will receive a new share

based on the new $t-1$ polynomial. It is assumed that each shareholder remembers his/her old share.

In addition, it is assumed that each shareholder has received one of the n secret shares created using Shamir's secret-sharing scheme [38] and that all honest shareholders perform the renewal protocol at the beginning of each time period as follows.

1. Each i'th shareholder $i \in [1..n]$ picks randomly $t-1$ numbers from a finite field, which define a polynomial $P_i(x)$ of degree $t-1$ whose free coefficient is zero $(P_i(0)=0)$.
2. Each i'th shareholder distributes the shares of $P_i(x)$ using the Verifiable Secret-Sharing scheme among the shareholders.
3. Each i'th shareholder receives the following shares: $P_1(i), ..., P_n(i)$, which include his own made share Pi(i)) and computes his/her new share $h(i)$ where $h(i) = f(i) + \sum_{c=1}^{n} P_c(i)$ ($f(i)$ is his/her old share.
4. Each i'th shareholder erases his/her old share $f(i)$.

4.4.8.2 Detection of Corrupted Shares

Every time the secret shares are renewed all participating shareholders have to make sure that the shares of the other shareholders have not been corrupted or lost. Since the lost or corruption of $n-(t-1)$ shares will cause the secret to be lost. This section presents a mechanism for detection of corrupted shares.

The main idea is to save a kind of fingerprint for each share. This fingerprint is common to all the shareholders, which allows the shareholders to periodically compare shares. Further, this protocol is based on the non-interactive VSS protocol described in Section 4.4.7.2. In addition to the non-interactive VSS protocol each shareholder stores the encryptions of all the shares he/she received from the other shareholders each time the key is renewed. This is achieved as follows:

- Perform the non-interactive VSS, so the encryption of the initial shares will be stored

at each shareholder.

- Using the homomorphic property, each i'th shareholder updates his/her set of encrypted shares by computing for every j using $E(h(i)) = E(f(i)) * \prod_{m=1}^{n} E(P_m(j))$. Note: This is only done with non-corrupted updated shares.

4.4.8.3 Reconstruction of Lost or Corrupted Shares

Once the Proactive Secret-Sharing protocol has discovered that a share has been corrupted or lost, it is necessary to reconstruct the share in order to ensure that no $n - (t-1)$ shares get lost. This has to be done without the involvement of the original dealer, since it may be possible that he/she is not available anymore. The idea behind this protocol is to provide a shareholder r whose share has been corrupted with information that will enable him/her to recover his/her share. One way would be to let each shareholder send his/her own share to r, which allows r to recover the polynomial $f(x)$, and then substitute r and recover his lost share $f(r)$. However, this would expose the secret S to r.

An algorithm, which does not allow r to construct the secret, but gives enough information to recover r's share is as follows.

1. Each i'th shareholder $(i \in [1...r-1], [r+1, n])$ chooses randomly a polynomial $P_i(x)$ of degree $t-1$ with where $P_i(r) = 0$ and $P_i(0) \neq 0$.
2. Each i'th shareholder (excluding the r'th shareholder) distributes shares of $P_i(x)$: $P_i(1)...P_i(n)$ using VSS, to all other shareholders excluding r.
3. Each i'th shareholder (excluding the r'th shareholder) receives $P_1(i), ..., P_{r-1}(i), P_{r+1}(i), ..., P_n(i)$ and calculates his/her new share for r: $h(i) = F(i) + \sum_{r-1}^{r-1}(i), P_{r+1}(i), ..., P_n(i)$ and sends it encrypted to r.
4. The r'th shareholder receives and decrypts these shares and interpolates them to recover $f(r)$. He/she receives a new polynomial in which the r'th share has the same value as the old lost share: $h(r) = f(r)$.

4.4.9 Evaluation of Secret-Sharing Schemes

Above we presented different secret-sharing schemes and their protocols. Starting with the Simple Sharing Control Schemes, which requires all shareholders to be available in order to reconstruct the secret. Due to the fact that our application is using a Peer-to-Peer network, this scheme is not suitable, since not all shareholders are necessarily online. In contrast, the (t,n)-threshold only requires t out of n shareholders to contribute their share to retrieve the secret, which gives us more flexibility. Shamir developed an ideal and perfect (t,n)-threshold scheme, which is used as basis for most of the other schemes such as the Verifiable Secret-Sharing scheme and the Proactive Secret-Sharing Scheme.

The Verifiable Secret-Sharing scheme provides advanced security by verifying the secret shares of the shareholder and dealer using either of two protocols, the interactive and the non-interactive. Both of the protocols require a dealer, who exchanged the secret shares. In our Peer-to-Peer network it is not ensured that the dealer is available at the time when the secret key is needed. Since we are dealing with sensitive data that is needed to provide the patient with better care, it is necessary to retrieve the needed information as soon as possible. Therefore, it is not acceptable to wait until the dealer is available again. Since we assume that every caregiver who is involved with our application and the information exchange as sworn the Hippocratic oath, we assume that the caregiver would act in the well-being of the patient and not cheat by sending wrong key shares. However, in case a caregiver does send a wrong key share then the patient privacy is still not violated since the wrong retrieved secret key, would not decrypt the sensitive information. By requesting more key shares, it is possible to find the "faulty" caregiver, since only t key shares are needed to reconstruct the key and it is assumed that no more than $n - (t - 1)$ caregivers would send wrong key shares. We use digital signature and public key encryption during the key share exchange to ensure that the share has not been corrupted during the transfer.

The Proactive Secret-Sharing scheme complicates the corruption and stealing of the secret shares and the secret itself by renewing the secret shares. However, since it is based on the Verifiable secret-sharing Scheme it is not suitable for us either due to the above given

reasons.

From the secret-sharing schemes described, Shamir's (t,n)-threshold scheme is the most suitable for our purpose.

Chapter 5

Design and Implementation

This chapter describes the design and implementation of the CDAShip as introduced in Chapter 3. Further, this chapter starts off listing the constraints that occur, due to implementation choices. This is followed by a detailed introduction of the architecture of the CDAShip, its components and their interactions. The chapter finishes with a description of the structure of the exchanged documents.

5.1 Major Dependencies

In order to provide a platform independent, reusable component the CDAShip was developed in Java. More precisely, the CDAShip is designed as a Java Bean, which means that it provides standards compliant accessor methods to allow retrieving and changing of attribute values (e.g. encryption algorithm) by external sources, and that it is possible to save the component state by using the Java Serialization mechanism.

To provide encryption functionality the CDAShip uses the Java Cryptography Extension (JCE) [31] and the Java Cryptography Architecture (JCA) [33], which are integrated into the Java 2 SDK, v 1.4. Both, JCE and JCA, are separately available for earlier versions of the Java 2 SDK. Due to the use of these packages the encryption algorithms are limited to the algorithms supported by the JCE. These encryption algorithms include Blowfish, Digital Encryption Standard (DES) and Triple DES.

Furthermore, the XML Security package from Apache[1] is used to create the XML Signature, which is integrated into the XML document. Therefore, the signing algorithms are limited to the ones supported by the XML Security package.

The CDA documents are passed between components as Document Object Model[2] document objects using the package org.w3c.dom, which is integrated into the Java 2 SDK, v 1.4. In order to parse the documents and to create new documents the Xerces2 Java Parser[3] is used.

In addition the CDAShip depends on the Sun XACML policy evaluation implementation [43]. This implementation does not yet support all features of the XACML 2.0 specification, therefore the CDAShip is limited to the features that are provided by Sun's implementation.

5.2 Architectural and Component-level Design

5.2.1 Component Description

The CDAShip enables different organizations and caregivers to participate in a secure medical data exchange using a peer-to-peer network. It consists of the following eight components:

- Import/Export
- CDACipher
- Policy
- KeyManager
- KeyRepository

[1]The XML Security package from Apache is available at `http://xml.apache.org/security/Java/index.html`

[2]The Document Object Model (DOM) is a platform- and language-neutral interface, which allows programs and scripts to dynamically access and update the content, structure and style of documents. [42]

[3]Xerces2 Java Parser is available at `http://xml.apache.org/xerces2-j/`

- Transport
- Audit
- FileSystem
- Util

These components and their interfaces are described below. Their design diagrams and a description of their classes can be found in Appendix E. Further, Figure 5.1 shows the CDAShip components and their relationships. It is important to mention that the Import/Export component is the only non-abstract component, which has interfaces that are exposed outside the CDAShip. All other component interfaces are used by the CDAShip itself and not intended to be used by the CDAShip user.

5.2.1.1 Import/Export

This component imports CDA documents from a FileSystem and triggers encryption and decryption. It also retrieves CDA documents from the Transport component and triggers the document verification and document persistency. In addition it retrieves the key shares from the Transport component, and saves them. If the key shares have been requested in order to access a encrypted section, it triggers the decryption of the section using the key shares.

The interfaces are briefly described below. A description of the Import/Export component classes can be found in Appendix E.1.

IExport is the interface used to communicate with this component in order to export CDA documents, key shares or key share requests. It provides methods to send a document, to forward a document or to access an unauthorized section.

IImport is the interface used to trigger the waiting for a requested key share. All other importing methods are triggered by events, which the Transport component throws when a document arrives.

IDocumentInfo is an interface used for the communication between the Import/Export and Transport component. It contains the document as a string and the document name.

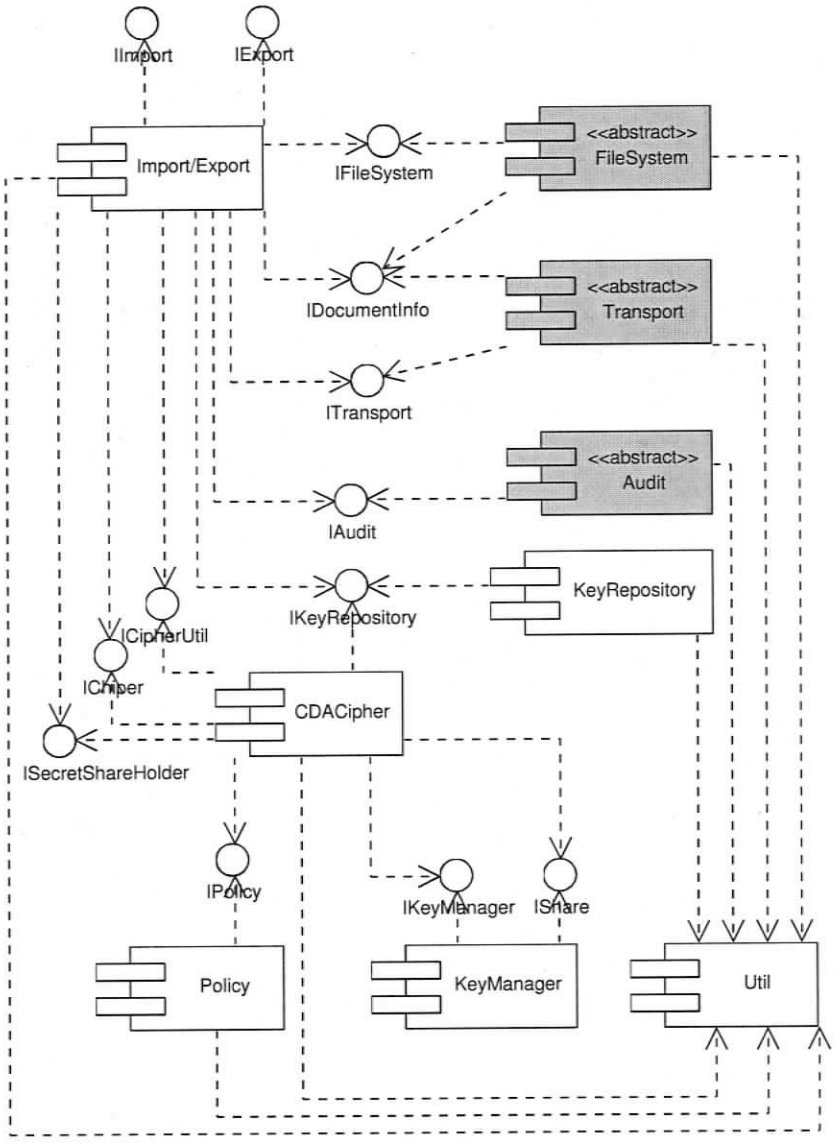

Figure 5.1. *CDAShip component diagram*

5.2.1.2 CDACipher

The CDAShip component is responsible for encrypting and decrypting using secret and public keys. It is also used for creating and verifying digital signatures.

It creates secret keys that are used for encryption, triggers the policy evaluation and encrypts the document and its sections according to the policy evaluation result with a given secret key. It also triggers the splitting of the secret key and creates and encrypts the key share information document.

The CDACipher component also decrypts given key share documents and triggers the reassembling of the secret key from the shares. Once the secret key is reassembled the CDACipher can decrypt the sections of a CDA document.

The interfaces of the CDACipher are:

ICipher is the main interface of the CDACipher and provides the CDACiphers functional methods such as encrypting a document or its section(s), signing it, or verifying a document's signature.

ICipherUtil provides accessor methods to the properties used by the CDACipher. These properties include the encryption algorithm, the message digest algorithm or the private key.

ISecretShareHolder provides accessor methods to the properties of a shareholder such as the ID of the shareholder or the location of the shareholder's public key.

A description of the classes of the CDACipher can be found in Appendix E.2

5.2.1.3 Policy

The Policy component automatically generates an XACML request from the given CDA document and the information about the document recipient. It further loads the XACML policies from a given location and evaluates them in order to determine whether the access to a document section or the whole document is granted or not.

Sun Microsystems has already released an open source Java implementation of the XACML standard [32], which is able to evaluate valid XACML policies. Our policy component uses Sun's implementation for the policy evaluation.

The Policy component has only one interface IPolicy, which is used by the CDACipher to trigger a policy evaluation and retrieve the evaluation result. The classes of the Policy component are described in Appendix E.3.

5.2.1.4 KeyManager

The KeyManager component creates key shares from a given secret key according to Shamir's secret-sharing scheme. It also assembles a secret key out of the given key shares assuming that enough key shares are presented.

The interfaces of the KeyManager component are described below. A description of classes of the KeyManager component can be found in Appendix E.4.

IKeyManager is the interface used by the CDACipher to communicate with the KeyManager component when splitting or recovering a secret.

IShare is the interface that provides information of a single share such as the threshold value, the total amount of shares of the original secret key or the expiry date.

5.2.1.5 KeyRepository

The KeyRepository component is responsible for loading public and private keys from a given URL string. By default it is possible to load PGP keys as well as keys that use the X.509 certificate. However, by adding a new Java provider to the component it can easily be extended to create other keys that use different certificates.

The KeyRepository has only one interface, IKeyRepository, which is used to communicate to the KeyRepository component. It contains methods to load private and public PGP keys and X.509 certificates. It also allows adding other certificate types. A description of

the classes of the KeyRepsoitory component can be found in Appendix E.5.

5.2.1.6 Transport

The Transport component enables the sending and receiving of the CDA documents, the key shares and key share requests over the network. This is an abstract component only consisting of an interface and events used to build a bridge between an of-the-shelf component, which is able to transfer data over the network such an email client, and the CDAShip.

5.2.1.7 Audit

The Audit component is an abstract component, which has to be implemented by the user of the CDAShip. It is used to store the audit data, which is produced whenever a key share is requested, into a database. It only consists of an interface providing an auditKeyShareRequest method.

5.2.1.8 FileSystem

The FileSystem component defines where the CDA documents, the policies as well as key shares are stored locally. It is also responsible for loading the key shares and policies according to their ID and URI, respectively. This component is an abstract component and only consists of an interface.

5.2.1.9 Util

The Util component includes classes that are used by most of the CDAShip components. These classes are mainly exceptions that are thrown during the processing of the documents, for example while ciphering or signing documents. The classes and exceptions of the Util component are described in Appendix E.9.

5.2.2 Component Interaction

5.2.2.1 Signing a document

The signing of a document and the encryption of the signature is a procedure that occurs often while exchanging documents using the CDAShip. Therewith the component interaction for signing a document is described here in detail.

The Import/Export component starts with passing the document to the CDACipher component and triggers the signing of the document by calling the signDoc() method. The CDACipher then signs the document using the sender's private key. The private key was specified at the time when the CDAShip was initialized. After the document is signed the signature needs to be encrypted using a secret key, which then will be encrypted using the recipient's public key. In order to retrieve the public key the Import/Export component passes the public key location and type, as well as the recipient's ID to the KeyRepository component, which then loads the public key and returns it to the Import/Export component.

Further, the signature has to be encrypted. Therefore, the Import/Export component passes the public key and the recipient's ID to the CDACipher component. The CDACipher component first of all creates a new secret key and generates the signature key information document, which contains the secret key in encrypted form[4]. Further, the CDACipher finalizes the signature encryption by ciphering the signature with the created secret key.

If it is a CDA document that has to be signed, then the policies that apply to the document will be signed and encrypted as well. This is done at this point, because of the intent to use only one secret key for the encryption of all signatures that are sent at one time. After the policies are also signed the CDACipher returns the signature key information document to the Import/Export component.

In the end the Import/Export component requests the signed document from the CDACipher.

Figure 5.2 illustrates this scenario.

[4]The structure of the signature key information document is explained in Section 5.3.2.3.

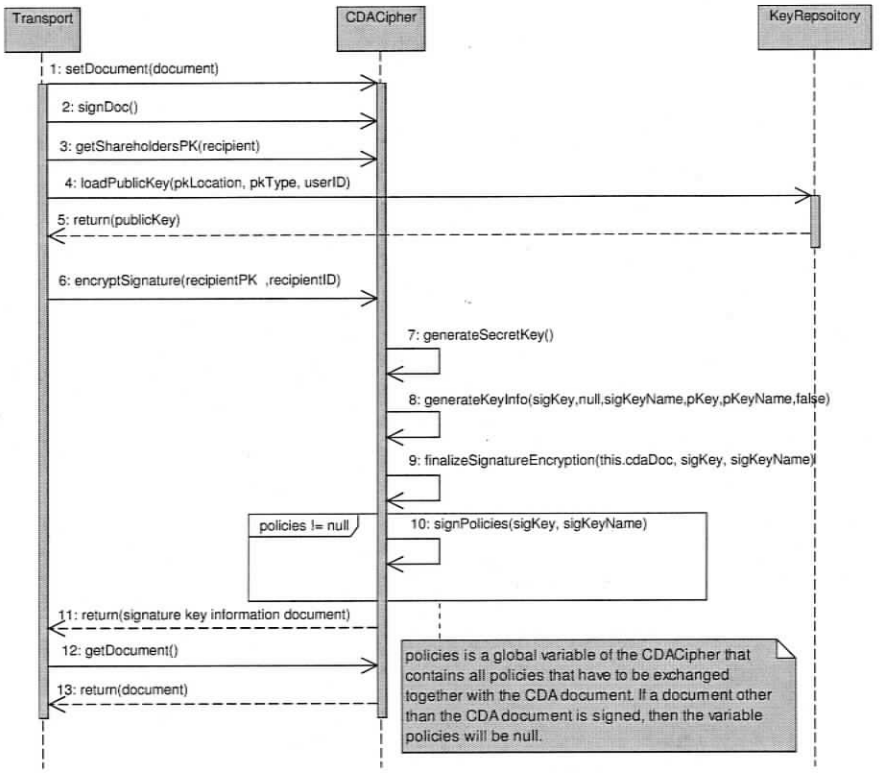

Figure 5.2. *Sequence Diagram: Signing of a document and encryption of the signature*

5.2.2.2 Verifying a signature

In order to verify the digital signature the sender's public key is needed. Therefore, the Import/Export component passes the location and type of the sender's public key as well as the sender's ID to the KeyRepository component, which then loads the sender's public key. Further, the Import/Export component sends the received document and the signature key information to the CDACipher and triggers the decryption of the signature. Once the signature is decrypted the Import/Export component triggers the signature verification by passing the sender's public key to the CDACipher component, which will return a boolean variable indicating whether the signature is valid or not.

Figure 5.3 shows this interaction in a sequence diagram.

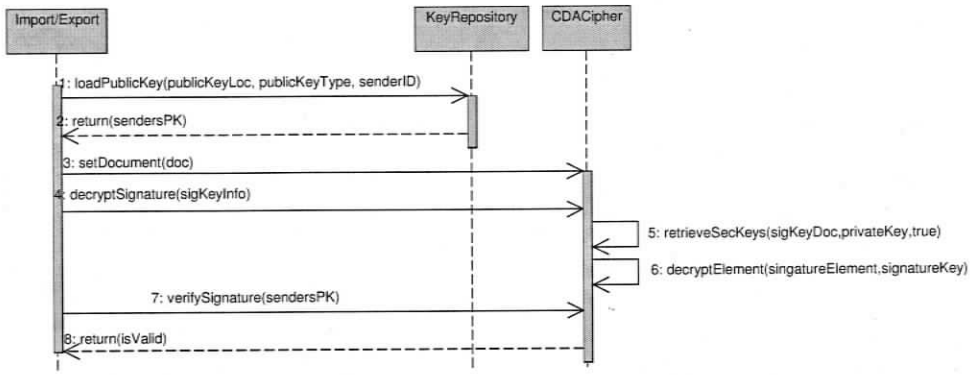

Figure 5.3. *Sequence Diagram: Verifying digital signature*

5.2.2.3 Sending of a CDA document

In order to send a secured CDA document from one caregiver to another the Import/Export component has to be provided with the CDA document, information of the recipient, a list of shareholders, who will receive key shares in case parts of the document have to be encrypted, as well as a list of polices that have to be evaluated prior to the exchange. The Import/Export component passes these to the CDACipher, which sends the document, the list of polices and the recipient information to the Policy component in order to evaluate the polices. After all policies have been evaluated and all obligations have been determined the Policy component returns the response to the CDACipher component. The response is processed by the CDACipher. If the evaluation result includes obligations stating that certain document sections have to be encrypted then a new secret key will be created and passed to the KeyManager component, which splits the key into n pieces[5]. After the secret key has been split and the shares have been returned to the CDACipher, it will process the obligations and encrypt each unauthorized section using the generated secret key. Further, the CDACipher returns a boolean variable to the Import/Export component indicating whether the CDA document includes encryption or not. This is needed later on in order to

[5]The details about how many shares to create, the threshold value, the type of encryption algorithm to be used, have to be set when initializing the CDAShip.

determine whether key shares have to be sent to the shareholders or not.

The Import/Export component now loads all policies from the FileSystem, since they also have to be exchanged together with the CDA document in case the recipient wants to forward the document later on. The policy documents are passed to the CDACipher. Further, the Import/Export component triggers the signing of the CDA document and the encryption of the digital signature by calling the corresponding methods of the CDACipher as it was described in Section 5.2.2.1. After all signatures have been encrypted the Import/Export component retrieves the signed polices from the CDACipher and passes them as well as the secured document, the information about the key that was used to encrypt the signature, the signed policies and the recipient's address to the Transport component. The Transport component then sends the the secured CDA document, the policies and the signature key information to the recipient.

In case the secured CDA document includes any encrypted sections and key shares have been created, the Import/Export component triggers the signing of the key share documents and passes the signed key shares together with the signature key information and the shareholders address to the Transport component, which sends out the key shares.

Figure 5.4 shows this interaction in form of a sequence diagram.

5.2.2.4 Receiving of a CDA or KeyShare document

During the initialization of the CDAShip the Import/Export component registers with the Transport component as DocumentReceivedEventListener. Whenever the Transport component receives a CDA document or a KeyShare document, throws a DocumentReceivedEvent and notifies all listeners by calling the documentReceived() method and passing a DocumentReceivedEvent object. The DocumentReceivedEvent object contains all information needed to process the received document.

The first step of the Import/Export component is triggering the deletion of the received message in the Transport component. This is necessary, because the CDAShip is supposed to handle the import of documents without the knowledge of the recipient. In case the

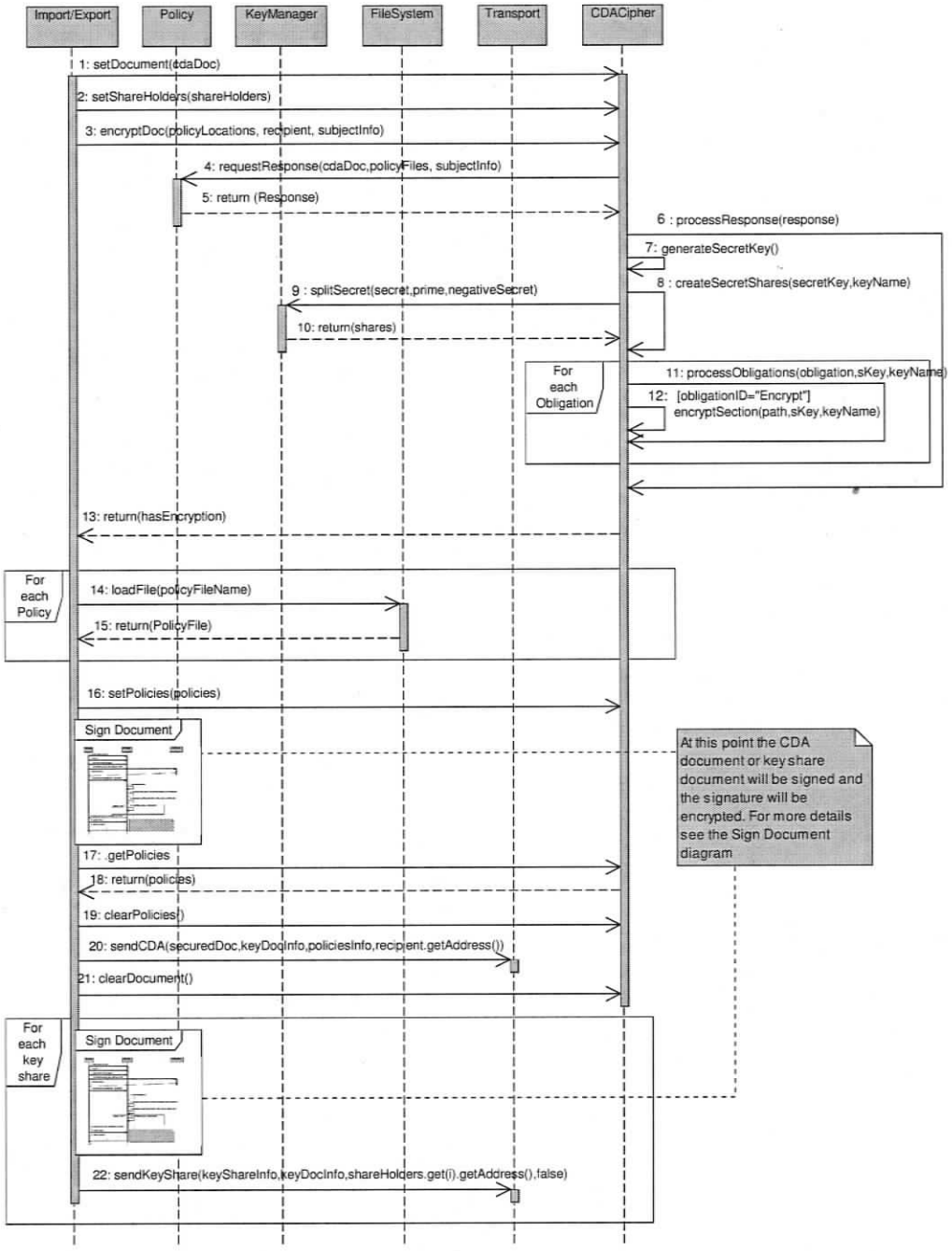

Figure 5.4. *Sequence Diagram: Encrypting and Sending of CDA documents*

recipient wants to be notified, the abstract FileSystem component can be designed to alert the recipient once the document is saved. But in general caregivers do not want their workflow interrupted by the arrival of a keyshare.

After the message has been deleted, the Import/Export triggers the verification of the digital signature as described in Section 5.2.2.2. If the signature verification is successful the Import/Export component will request the document from the CDACipher and remove the signature from the document, since it is no longer needed.

If the received document was a CDA document then the DocumentReceivedEvent, which was passed to the Import/Export component initially, also includes a list of policies. The signatures of the policies will be decrypted and verified in the same way as it was done for the document. Further, the polices will be passed to the FileSystem, which will save them. Finally, the CDA document will also be passed to the FileSystem in order to save it. The recipient may now be notified by the FileSystem that a CDA document has been received.

In case the received document was a KeyShare document, it will be passed to the FileSystem and therewith saved.

Figure 5.5 shows this interaction in a sequence diagram.

5.2.2.5 Receiving of a key share request

In addition to the DocumentReceivedEventListener the Import/Export component also registers as a KeyShareRequestReceivedEventListener with the Transport component. If the Transport component receives a key share request, it throws a KeyShareRequestReceivedEvent event and therewith notifies the Import/Export component. The KeyShareRequestReceivedEvent contains the request document, the signature key document, the ID of the requested key share and other information needed to process the key share request.

Before the key share request will be processed, the Import/Export component triggers the deletion of the message that arrived at the Transport component. Further, it will verify the digital signature of the request document as it was described in Section 5.2.2.2.

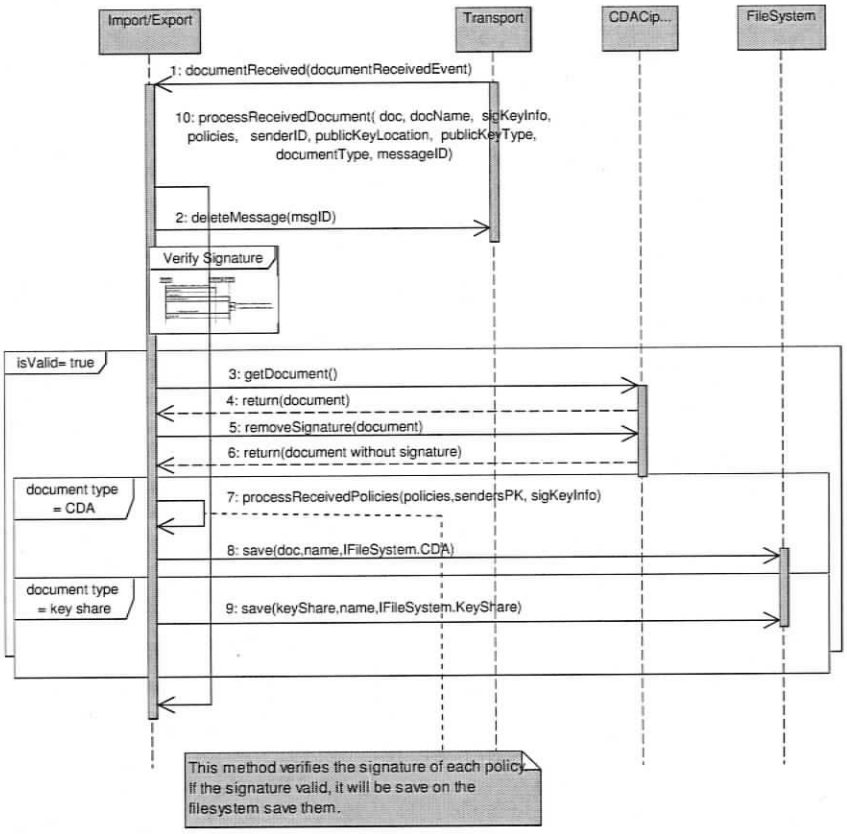

Figure 5.5. *Sequence Diagram: Processing received CDA document or key share*

If the signature verification was successful, the Import/Export component will resume with retrieving the key share document from the FileSystem component by passing the ID of the key share. Further, the Import/Export component will pass the sender's public key location and type as well as the sender's ID to the KeyRepository in order to load the sender's public key.

Since the key share value in the key share document is encrypted with the shareholder's public key, it is necessary to decrypt the key share value and encrypted it again with the requester's public key. The re-encryption is done by the CDACipher. Once the re-encrypted key share document has been returned to the Import/Export component, it will sign the key share document as described in Section 5.2.2.1. After the document has been signed,

it will be passed together with the signature key information and the requester's ID to the Transport component, which then sends the key share to the requester. Finally, the Import/Export component will delete the document from the CDACipher.

Figure 5.6 shows this scenario in detail.

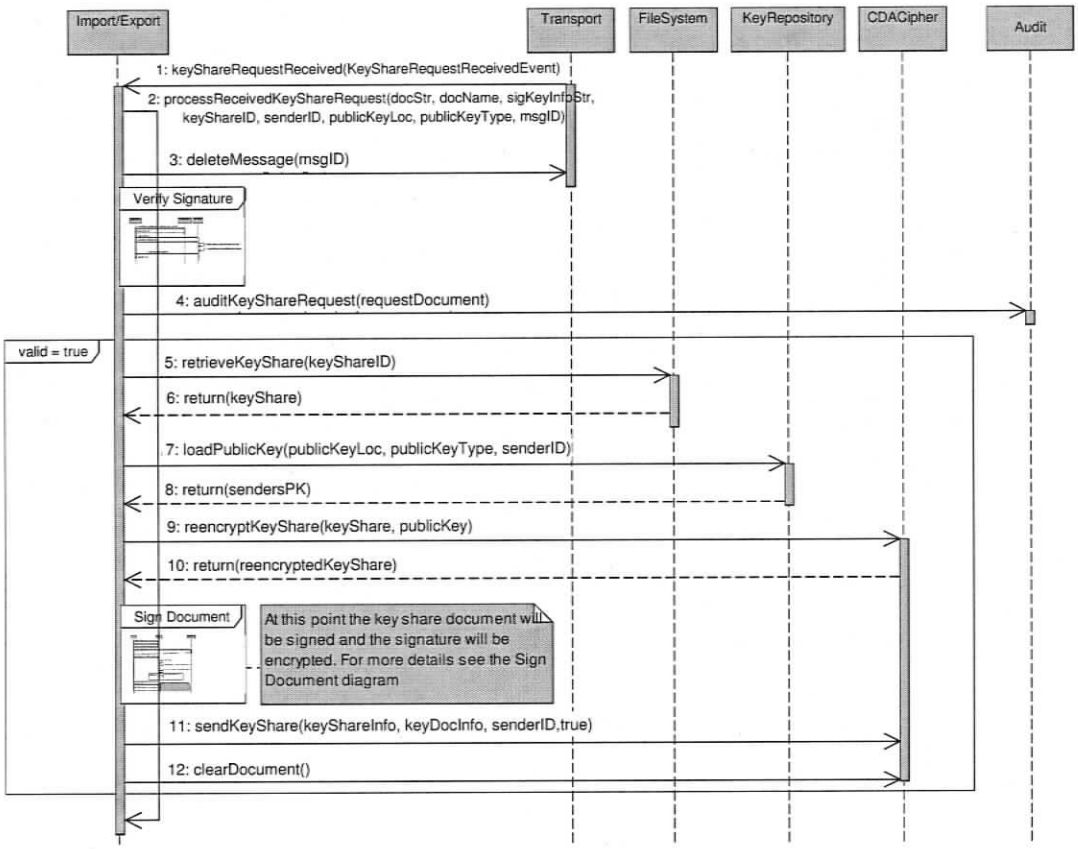

Figure 5.6. *Sequence Diagram: Processing received key share request*

5.2.2.6 Accessing an unauthorized section

In case an encrypted section needs to be accessed, the accessSection method from the Import/Export component has to be called. The method requires the following information

- the requester
- the reason why the section has to be accessed
- the document, which includes the section
- the document ID
- the xpath expression to the encrypted section
- a list of all policy files that have to be evaluated
- the information about the requester, which needs to be added to the policy request
- patient's address in order to notify the patient.

The accessSection method only passes this information further to a private method also called accessSection, which has one additional parameter that includes the properties in case a document is forwarded. These properties include amongst other the information about the caregiver to whom the document should be forwarded to and who should receive a key share in case the document needs to be encrypted prior forwarding. However, since this document is not forwarded this parameter will be null. The next steps are all done within the called accessSection method.

After the private accessSection method has been called, the Import/Export component sends the CDA document to the Cipher component and calls the evaluatePolicies method by passing the list of policy files as well as the subject information needed for the request. The CDACipher then passes these to the Policy component, which generates the request, evaluates the policies and returns the policy response to the CDACipher. The response will be passed on to the Import/Export component. If according to new the policy evaluation the access to the section is still not granted and if the access is not needed in order to forward a document then the Import/Export component will return a generated and unique request ID. This request ID will be stored internally at the Import/Export component. If the caregiver insists on accessing the section against the policies then an InsistRequestEvent has to be thrown. The InsistRequestEvent has to contain the returned request ID, which ensures that the Import/Export component knows which request to process. Since the request ID

was generated by the Import/Export component it ensures that request cannot be faked. In case the policy evaluation resulted in the fact that the access is granted or the document is supposed to be forwarded, the Import/Export component will throw the corresponding InsistRequestEvent event. Figure 5.7 illustrates this scenario.

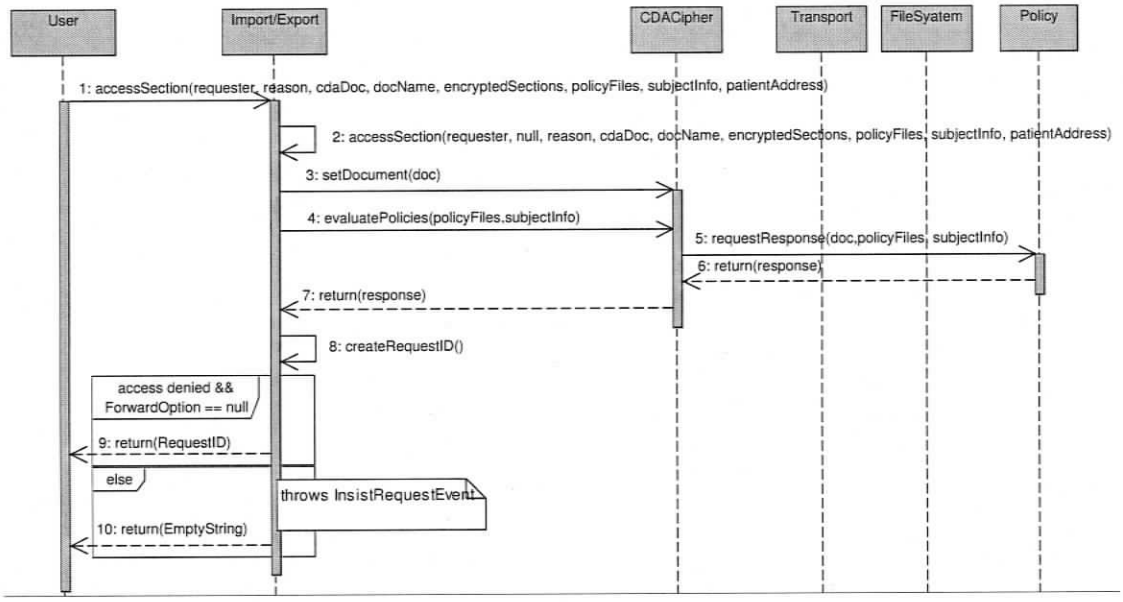

Figure 5.7. *Sequence Diagram: Accessing a encrypted section*

After a InsistRequestEvent has been thrown, the Import/Export gets the request ID from a list and processes by retrieving the ID(s) of the secret key(s) that were used to encrypt the section(s) from the CDA document. Further, for each secret key ID the Import/Export component generates a request document, which is then passed to the CDACipher component in order to sign the request. After the request has been signed it will be passed together with the reason and the patient address to the Transport component, which then notifies the patient about the key share request.

Additionally the Import/Export component requests the key share information document from the FileSystem component by passing the key share ID. The key share information document contains the information about all other shareholders as well as information needed to reconstruct the secret key. The Import/Export component extracts each share-

holder's information from the key share information document. Further, for each shareholder the Import/Export component will sign the key share request and encrypt the signature as described in Section 5.2.2.1. Once the key share request is signed and encrypted, the Import/Export component passes the key share request document, the signature key document, the key share ID as well as the address of the shareholder to the Transport component, which then sends off the document.

After all key share requests have been distributed the Import/Export component clears the document that is still stored in the CDACipher. Further, it will call the waitForRequestedKS method, which adds the information about the requested key share to a list in order to identify the requested key shares once they arrive.

The steps, beginning from receiving the InsistRequestEvent are illustrated in Figure 5.8

Besides the registration with the KeyShareRequestReceivedEventListener and DocumentReceivedEventListener, the Import/Export component also registers with the RequestedKeyShareReceivedEventListener from the Transport component. The RequestedKeyShareReceivedEvent is thrown by the Transport component whenever a requested key share arrives. This will notify the Import/Export component, which then processes the requested key share. All information needed to process the key share is included in the RequestedKeyShareEvent.

In order to process the key share, the Import/Export triggers the deletion of the message at the Transport component. After this it will verify the signature of the key share document as described in 5.2.2.2.

In case the signature validation was successful, the Import/Export component requests the document from the CDACipher and removes its signature, since it is no longer needed. Once the signature is removed the key share document will be added to the list of key shares that have already been received.

If the Import/Export component has already received enough key shares to reconstruct the secret key, then it will call the method enoughKeySharesReceived and pass on the key share information documents that have been collected. This information includes the re-

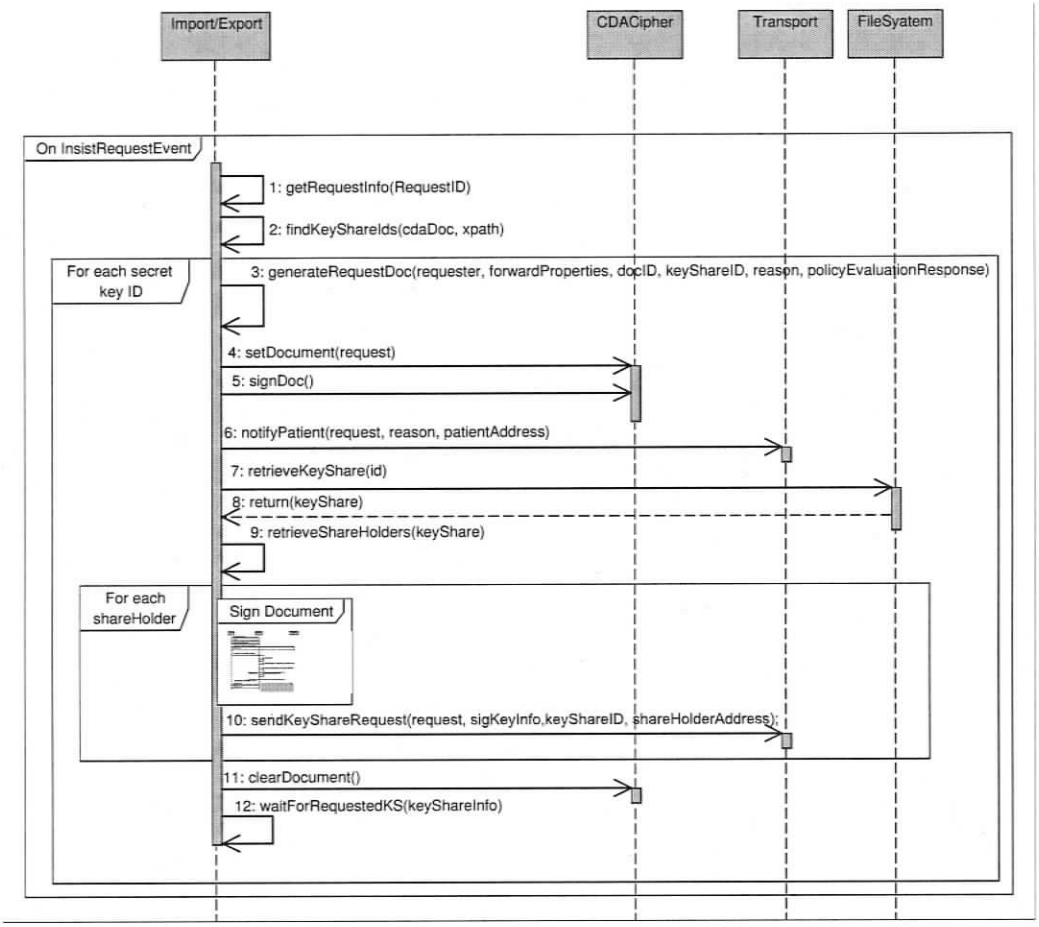

Figure 5.8. *Sequence Diagram: Continuing accessing section on InsistRequestEvent*

ceived key shares, the CDA document with the encrypted sections, the path to the encrypted sections as well as an indication stating whether the key shares were requested in order to forward the document or not.

Further it will pass the received key share documents to the CDACipher component, which passes them further to the KeyManager in order to recover the secret. The KeyManager returns the secret as a BigInteger variable and CDACipher will create the secret key out of the BigInteger variable from which the CDACipher creates the secret key that is returned. Afterwards the Import/Export component passes the CDA document to the CDACipher and triggers the decryption for each section that has to be decrypted. As a

result of each decryption the CDACipher will return a boolean variable indicating whether the decryption was successful or not. If all decryptions were successful, the Import/Export component will request the decrypted document from the CDACipher.

In case the document was decrypted in order to forward it to another caregiver, the Import/Export component calls the sendCDA method and proceeds as described in 5.2.2.3.

If the document was decrypted, because the encrypted information was needed locally, then the document will be passed to the FileSystem in order to save it. The FileSystem may notify the caregiver that the document has been decrypted.

Finally, the Import/Export component deletes the document from the CDACipher component.

Figure 5.9 illustrates this scenario.

5.2.2.7 Forwarding a CDA document with encrypted sections

In case an document needs to be forwarded to another caregiver, the forwardDocument method from the Import/Export component has to be called. The method requires the following information

- the forwarder
- the reason why the document has to be forwarded
- the document
- the document ID
- a list of all policy files that have to be evaluated
- the information about the recipient, which needs to be added to the policy request
- patient's address in order to notify the patient
- the recipient's public key, email etc.
- list of the shareholders who will receive a key share in case the document has to be encrypted again.

The forwardDocument scans the document for all encrypted sections and collects the xpath expressions to these sections in an array. Further, it creates a ForwardProperty ob-

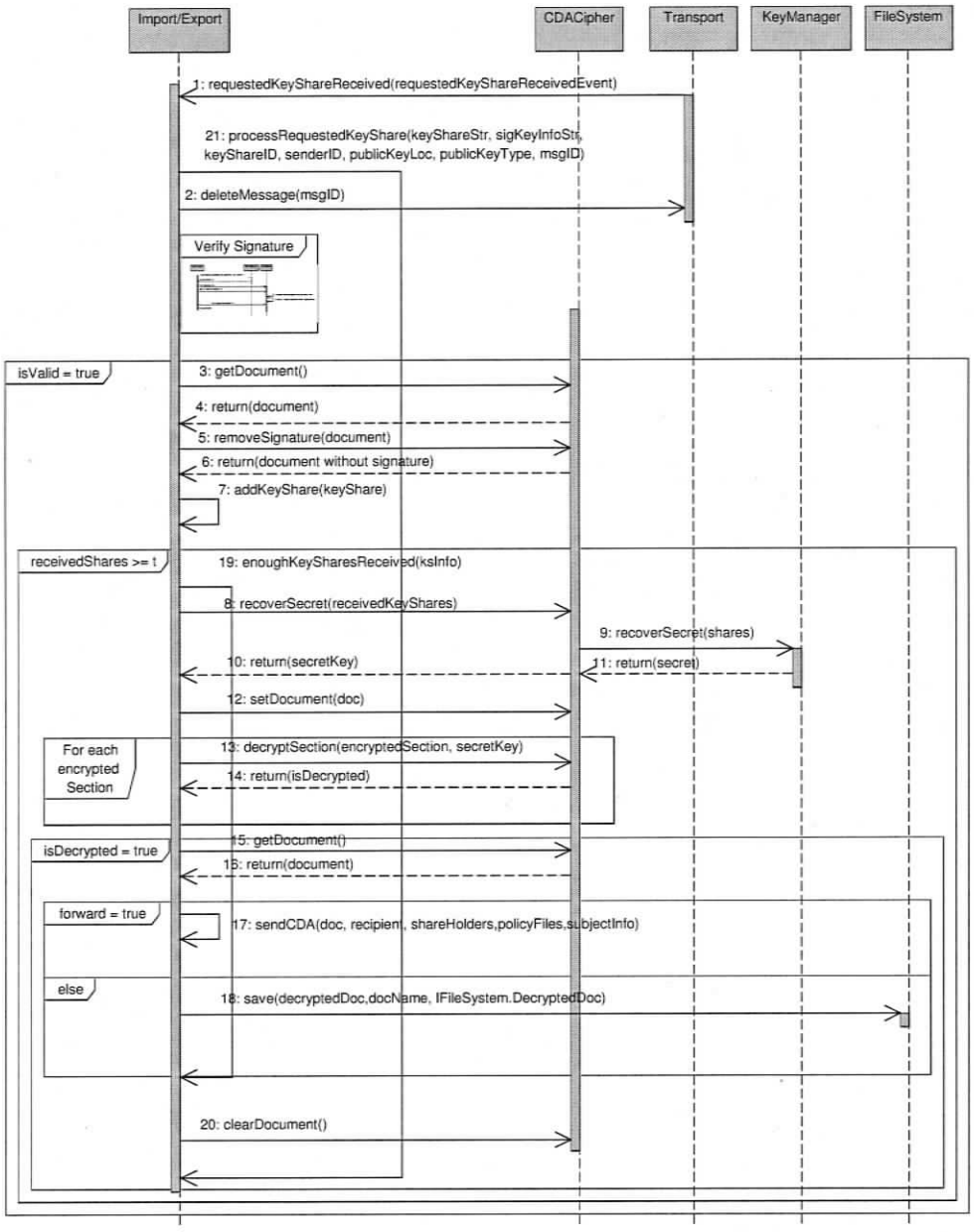

Figure 5.9. *Sequence Diagram: Processing requested key share*

ject, which contains all information needed to forward the document later. This information includes the properties of the recipient, the list of the shareholders and the policies. Afterwards the information, which was passed to the forwardDocument method, as well as the array with the xpath expressions and the ForwardProperty object are passed to private accessSection method, which notifies the patient, creates the key share requests and triggers the waiting for the requested key shares. Detailed information about this process can be found in Section 5.2.2.6.

5.3 Document Structures

As mentioned in Chapters 2 and 3 the CDAShip uses the HL7's Clinical Document Architecture (CDA) as format for information exchange and the XACML specification for controlling access to the CDA documents and their sections. Both standards, CDA as well as XACML, are based on XML schemas and documents. Therefore, it is self-evident to use XML as representation for all documents that are exchanged using the CDAShip. This section describes the structure of the documents used and created by the CDAShip.

5.3.1 Encryption and Signature

Since it cannot be assumed that each participant in the medical information exchange uses the introduced CDAShip adapter component it is necessary that the secured CDA documents, which the CDAShip produces are compliant with other technologies that handle valid CDA documents. Therefore, it is required that the context-based encrypted CDA documents are still valid CDA documents. Due to the fact that CDA documents are represented using XML, the CDACipher will encrypt and sign the CDA documents according to the W3C specifications for XML Encryption [40] and XML Signature [28], respectively. This section describes how these specifications are applied in order to provide context-based security for CDA documents.

5.3.1.1 Encryption Structure of Elements

One key goal of the CDAShip component is that it is able to encrypt unauthorized sections using a secret key. In this approach, encrypting a section or in XML terminology encrypting an element is defined as encrypting the content of the element, which means that the element tag is still present after enccyption. This is necessary to ensure that the document conforms to the CDA structure after encryption. The encrypted document now complies with the CDA Level One [12] document type, which contains the clinical data as one narrative node. The W3C XML Encryption Syntax and Processing [40] recommends the following structure for the encrypted data.[6]

```
1  <!ELEMENT EncryptedData (EncryptionMethod?,ds:KeyInfo?,CipherData,
                            EncryptionProperties?)>
2  <!ATTLIST EncryptedData
       Id ID #IMPLIED
       Type CDATA #IMPLIED
       MineType CDATA #IMPLIED
       Encoding CDATA #IMPLIED>

4  <!ELEMENT EncryptionMethod EMPTY>
5  <!ATTLIST EncryptionMethod Algorithm CDATA #REQUIRED>

7  <!ELEMENT ds:KeyInfo (#PCDATA | ds:KeyName | ds:KeyValue |
                         ds:RetrievalMethod | ds:X509Data |
                         ds:PGPData | ds:SPKIData | ds:MgmtData)*>

9  <!ELEMENT CipherData (CipherValue | CipherReference)>
10 <!ATTLIST CipherReference URI CDATA #IMPLIED>
```

The element <EncryptedKey> contains information about the key used to encrypt this element. It is not planned to encrypt the secret key at the same time as the element is encrypted, since the key still needs to be split into the key shares. Therefore, the <EncryptedKey> element is not relevant at the time of encryption, rather it is used in the document which contains the secret share key information. More information can be found in Section 5.3.2.1.

[6]The complete schema of the W3C XML Encryption including the namespaces as well as elements that have not been defined here can be found at `http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd`

The element <CipherValue> contains the encrypted data value. The W3C XML Encryption recommendation envisions that the element content has to be serialized in UTF-8 before it can be encrypted. Additionally, it specifies that the ciphered value has to be base64 encoded.

The other elements are self-explanatory or not relevant at this state of the project. Additional information is available in the W3C XML Encryption recommendation [40].

Since the CDAShip project is in the initial stage, the first prototype represents the encrypted data in a simplified structure as follows.

```
1  <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
2      Type="http://www.w3.org/2001/04/xmlenc#Element">
3      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
          blowfish"/>
4      <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
5          <ds:KeyName>RIwHnKTomxZivWUUgiEYQQ==</ds:KeyName>
6      </ds:KeyInfo>
7      <CipherData>
8          <CipherValue>QSYp03wa3</CipherValue>
9      </CipherData>
10 </EncryptedData>
```

5.3.1.2 Encryption Structure of Attributes

An XML element can contain attributes which are included in the element tag. Due to the fact that encrypting an element means encrypting the content and not the element tag itself, the attribute values of an element will not be encrypted. Furthermore, the W3C XML Encryption [40] does not specify a structure how to encrypt attributes. However, in order to provide a secure transmission it is necessary to cipher section attributes too. Ed Simon, one of the authors of the W3C XML Encryption, proposes [39] a method for encrypting XML attributes. Since it is possible that this proposal will be included within the next version of the W3C XML Encryption, the CDAShip adopts his idea.

Ed Simon proposes replacing the attribute value with its cipher value and adding an EncryptedDataManifest attribute to the element. This added attribute will indicate that the attributes are encrypted. Furthermore, an <EncryptedDataManifest> element is added as

a child node to the element. Included in the <EncryptedDataManifest> element is the necessary information to decrypt the ciphered attribute data. For each encrypted attribute an <EncryptedData> element is added to the <EncryptedDataManifest>, which has the same structure as described in 5.3.1.1. Additionally, each <EncryptedData> element has an attribute called Name, which contains the name of the attribute whose information is stored in the <EncryptedData> element. This will make it possible to retrieve the needed information for decryption. The structure used for the encrypted attributes is given below.

```
1  <document EncryptedDataManifest="./EncryptedDataManifest"
2      id="Replace value with cipher value">
3      <EncryptedDataManifest>
4          <EncryptedData Name="id" Type="AttributeValue">
5              <EncryptionMethod/>
6              <KeyInfo>
7                  <KeyName/>
8              </KeyInfo>
9          </EncryptedData>
10     </EncryptedDataManifest>
11 </document>
```

5.3.1.3 Signature Structure

The W3C XML Signature recommendation [28] defines the syntax and processing rules for XML Signatures. It distinguishes between enveloped and detached XML Signatures. Enveloped signatures sign the document which contains it, whereas detached signatures sign external data. This project uses enveloped signatures for all documents that are exchanged using the CDAShip, since it seems to be more appropriate to transfer a document, which includes the signature rather than a document and the signature separately. The signature, which will be attached at the bottom of the CDA document, has the following structure suggested by [28].

```
1  <!ELEMENT SignedInfo (CanonicalizationMethod, SignatureMethod,
                         Reference+)  >
2  <!ATTLIST SignedInfo Id ID #IMPLIED >

4  <!ELEMENT CanonicalizationMethod (#PCDATA )* >
5  <!ATTLIST CanonicalizationMethod Algorithm CDATA #REQUIRED >
```

```
 7 <!ELEMENT SignatureMethod (#PCDATA)* >
 8 <!ATTLIST SignatureMethod Algorithm CDATA #REQUIRED >

10 <!ELEMENT Reference (Transforms?, DigestMethod, DigestValue)  >
11 <!ATTLIST Reference
      Id  ID  #IMPLIED
      URI CDATA   #IMPLIED
      Type CDATA #IMPLIED>

13 <!ELEMENT DigestMethod (#PCDATA)* >
14 <!ATTLIST DigestMethod  Algorithm CDATA #REQUIRED >

16 <!ELEMENT DigestValue  (#PCDATA)  >

18 <!ELEMENT KeyInfo (#PCDATA | KeyName | KeyValue | RetrievalMethod
                    | X509Data | PGPData | SPKIData | MgmtData)* >
19 <!ATTLIST KeyInfo Id ID #IMPLIED >
```

The following list briefly explains the major elements of the XML Signature, more information can be found at [28].

<**SignedInfo**> contains information about the actual signed data.

<**CanonicalizationMethod**> contains the algorithm used to canonicalize the signed information before it is digested. A detailed description of canonicalization can be found in [10].

<**SignatureMethod**> specifies the algorithm used to sign the data.

<**Reference**> includes digest method and digest values from the data, which is used for signature verification.

<**Transforms**> gives information about the transformation method, which has been applied to the data before creating the message digest.

<**DigestMethod**> contains the method used to create the message digest.

<**DigestValue**> is the value of the message digest.

<**SignatureValue**> gives the value of the signature.

<**KeyInfo**> includes information about the key used for signing.

The CDACipher component will create digital signatures according to the W3C XML Signature recommendation.

5.3.2 Key Information

This section includes the description of the document structures for the key shares, the key share request as well as the structure for the signature key information document, which contains the secret key used to encrypt the digital signature.

5.3.2.1 Key Share Structure

The W3C XML Encryption [40] recommendation specifies a structure for transmitting encrypted keys together with the encrypted data. The key information is included in the <EncryptedData> element that was described in Section 5.3.1.1.

However, since the secret key still has to be split into n pieces according to Shamir's (t,n)-threshold scheme (see Section 4.4.5) and since the generated key shares may be updated, the CDAShip component does not include the secret key information together with the <EncryptedData> element. Another option could be that the key information is added later to the <EncryptedData> element, but this is also not possible in the CDAShip approach, because an element, which contains an encrypted subsection can also be encrypted and therewith it is not possible to add the key information to the encrypted subsection, without decrypting the parent section. For these reasons the CDAShip component creates a new XML document including the information about the secret key shares. In addition to the encrypted key share, the document will also include the contact information of the other shareholders as well as other information needed to reconstruct the secret key.

The KeyShare document structure is shown below. Its XML schema definition can be found in Appendix A.

```
1 <!ELEMENT KeyShare (xenc:EncryptedKey, KeyName,
     SecretKeyEncryptionMethod, KeyShareInfo)>
2 <!ATTLIST KeyShare
3     xmlns CDATA #FIXED 'http://www.ppci.ca/CDAShip/KeyShare#'
4     xmlns:ds CDATA #FIXED 'http://www.w3.org/2000/09/xmldsig#'
5     xmlns:xenc CDATA #FIXED 'http://www.w3.org/2001/04/xmlenc#'
6     xmlns:xsi CDATA #FIXED
        'http://www.w3.org/2001/XMLSchema-instance'
```

```
7       xsi:schemaLocation CDATA #FIXED
            'http://www.ppci.ca/CDAShip/KeyShare# http://www.cs.uvic.ca
            /~obry/CDAShip/Schema/KeyShare.xsd'
8       Id ID #REQUIRED>

10  <!ELEMENT xenc:EncryptedKey (xenc:EncryptionMethod, ds:KeyInfo,
        xenc:CipherData, xenc:CarriedKeyName)>

12  <!ELEMENT xenc:EncryptionMethod EMPTY>
13  <!ATTLIST xenc:EncryptionMethod Algorithm CDATA #REQUIRED>

15  <!ELEMENT ds:KeyInfo (ds:KeyName)>
16  <!ELEMENT ds:KeyName (#PCDATA)>

18  <!ELEMENT xenc:CipherData (xenc:CipherValue)>
19  <!ELEMENT xenc:CipherValue (#PCDATA)>

21  <!ELEMENT xenc:CarriedKeyName (#PCDATA)>

23  <!ELEMENT SecretKeyEncryptionMethod (#PCDATA)>

25  <!ELEMENT KeyShareInfo (ShareNumber, Threshold, TotalShares,
        Prime, NegativeSecret, ExpiryDate, ShareHolders)>

27  <!ELEMENT ShareNumber (#PCDATA)>
28  <!ELEMENT Threshold (#PCDATA)>
29  <!ELEMENT TotalShares (#PCDATA)>
30  <!ELEMENT Prime (#PCDATA)>
31  <!ELEMENT NegativeSecret (#PCDATA)>

33  <!ELEMENT ExpiryDate (Month, Day, Year)>
34  <!ELEMENT Month (#PCDATA)>
35  <!ELEMENT Day (#PCDATA)>
36  <!ELEMENT Year (#PCDATA)>

38  <!ELEMENT ShareHolders (ShareHolder+)>
39  <!ELEMENT ShareHolder (Address, ID, PublicKeyInfo)>
40  <!ELEMENT Address (#PCDATA)>
41  <!ELEMENT ID (#PCDATA)>

43  <!ELEMENT PublicKeyInfo (PublicKeyLocation, PublicKeyType)>
44  <!ELEMENT PublicKeyLocation (#PCDATA)>
45  <!ELEMENT PublicKeyType (#PCDATA)>
```

The following list briefly describes the child elements of the <KeyShare> element.

<**EncryptedKey**> its child element include the key share information according to W3C XML Encryption [40].

<**EncryptedMethod**> defines the method used to encrypt the secret key share.

<**KeyInfo**> its child element <KeyName> gives information about the name of the key used to encrypt the secret key share.

<**CipherData**> its child element <CipherValue> includes the base64 ciphered data of the secret key share.

<**CarriedKeyName**> is the name of the encrypted secret key.

<**KeyName**> is the name and id of the secret key to which this key share belongs to. The value of this element should be the same as the value given in <CarriedKeyName>. This replication is necessary, because a CDA document recipient does not necessarily have to receive a key share. In which case the <EncryptedKey> element and therewith the <CarriedKeyName> are omitted, but in order to request the key shares from other shareholders the recipient needs to know the key name.

<**SecretKeyEncryptionMethod**> specifies the encryption method which the secret key share is using[7].

<**ShareNumber**> indicates which of the n shares the shareholder received.

<**Threshold**> defines number of shares needed to reconstruct the secret key.

<**TotalShares**> indicates how many shares were created.

<**Prime**> contains the prime used to create the secret shares.

<**NegativeSecret**> indicates whether the secret is a negative value[8].

<**ExpiryDate**> specifies the key share life time.

<**ShareHolders**> includes a list of all shareholders.

<**Shareholder**> contains information such as address, id and public key data

[7]This element does not exist in the W3C specification. It was added in order to be able to regenerate the secret key shares.

[8]This is needed because in the implementation the secret key is transformed into a BigInteger before it will be split. This BigInteger value may be negative. However, it is not possible to spilt a negative value using the Shamir's (t,n)-threshold scheme. In case the value is negative the absolute value is split. When reconstructing the secret it is necessary to know whether the absolute value was used.

for a single shareholder.

5.3.2.2 KeyShareRequest Structure

In order to access an encrypted section the distributed key shares have to be requested from the shareholders. The KeyShareRequest document includes in addition to the requested key share id, information about who wants to access which document and for what reason. This information is needed for auditing purpose. The structure of the Request document is given and briefly explained below [9].

```
1  <!ELEMENT KeyShareRequest (Requester, RequestFor, Reason,
                              xacml-context:Response)>
2  <!ATTLIST KeyShareRequest
3      xmlns CDATA #FIXED
           'http://www.ppci.ca/CDAShip/KeyShareRequest#'
4      xmlns:ks CDATA #FIXED 'http://www.ppci.ca/CDAShip/KeyShare#'
5      xmlns:xacml-context CDATA #FIXED
           'context="urn:oasis:names:tc:xacml:2.0:context:schema:os'
6      xsi:schemaLocation CDATA #FIXED
           'http://www.ppci.ca/KeyShareRequest# http://www.cs.uvic.ca
           /~obry/CDAShip/KeyShareRequest.xsd'>

8  <!ELEMENT Requester (ks:ShareHolder)>
9  <!ELEMENT RequestFor (CDA_ID, KeyShareID)>
10 <!ELEMENT CDA_ID (#PCDATA)>
11 <!ELEMENT KeyShareID (#PCDATA)>
12 <!ELEMENT Reason (ReasonReason, ForwardTo?)>
13 <!ELEMENT ReasonReason (#PCDATA)>
14 <!ELEMENT ForwardTo (ks:ShareHolder)>
15 <!ELEMENT xacml-context:Response (xacml-context:Result+)>
```

<**Requester**> contains a <ShareHolder> element as described in 5.3.2.1, which includes the information about the requester.

<**RequestFor**> specifies the id of the CDA document and the id of the needed key share.

<**ReasonReason**> includes the reason why the key share is requested.

[9] *"ks" refers to the KeyShare document schema, which was discussed in 5.3.2.1 and defined in Appendix A and "xacml-context" refers to the xacml-context schema, which can be found in the XACML documentation [43]*

<**ForwardTo**> if the key share is needed in order to forward a CDA document then the <ForwardTo> is added and includes the shareholder information for the caregiver who the document will be forwarded to.

<**Response**> contains the response of the policy evaluation. A detailed description of this element is given in Section 5.3.3.3.

The XML schema definition of the KeyShareRequest document can be found in Appendix B.

5.3.2.3 Signature Key Information Structure

The CDAShip uses a digital signature (see Section 4.3.2.1) in order to authenticate a sender of the document. This digital signature is attached to the exchanged document as described in Section 5.3.1.3. Furthermore, the signature is encrypted in order to ensure that it cannot be modified during the data transfer. The secret key used to encrypt the signature is exchanged together with the documents in a so called SigKeyInfo document. To guarantee that the secret key will not be misused by somebody other than the intended recipient it is encrypted with the receiver's public key. Basically the SigKeyInfo document uses a subset of the elements used in the KeyShare document. The structure of the SigKeyInfo is illustrated below[10] and the XML schema definition can be found in Appendix A.

```
1  <!ELEMENT SigKeyInfo (xenc:EncryptedKey, KeyName,
                         SecretKeyEncryptionMethod)>
2  <!ATTLIST SigKeyInfo
3      xmlns CDATA #FIXED 'http://www.ppci.ca/CDAShip/KeyShare#'
4      xmlns:ds CDATA #FIXED 'http://www.w3.org/2000/09/xmldsig#'
5      xmlns:xenc CDATA #FIXED 'http://www.w3.org/2001/04/xmlenc#'
6      xmlns:xsi CDATA #FIXED
           'http://www.w3.org/2001/XMLSchema-instance'
7      xsi:schemaLocation CDATA #FIXED
           'http://www.ppci.ca/CDAShip/KeyShare# http://www.cs.uvic.ca
           /~obry/CDAShip/Schema/KeyShare.xsd'>

9  <!ELEMENT xenc:EncryptedKey (xenc:EncryptionMethod, ds:KeyInfo,
                                xenc:CipherData, xenc:CarriedKeyName)>
```

[10] *"xenc" refers to the restricted W3C XML Encryption Syntax described in Appendix C and "ds" refers to the restricted W3C XML Signature Syntax described in Appendix D*

```
11 <!ELEMENT xenc:EncryptionMethod EMPTY>
12 <!ATTLIST xenc:EncryptionMethod Algorithm CDATA #REQUIRED>

14 <!ELEMENT ds:KeyInfo (ds:KeyName)>
15 <!ELEMENT ds:KeyName (#PCDATA)>

17 <!ELEMENT xenc:CipherData (xenc:CipherValue)>
18 <!ELEMENT xenc:CipherValue (#PCDATA)>

20 <!ELEMENT xenc:CarriedKeyName (#PCDATA)>

22 <!ELEMENT KeyName (#PCDATA)>
23 <!ELEMENT SecretKeyEncryptionMethod (#PCDATA)>
```

5.3.3 XACML Policy, Request and Response Structure

As described in Section 4.2.1.3, XACML is the policy language that is the most appropriate for our purposes. This section describes the basic structure of the XACML policy, request and response and the requirements made to fit our approach. A detailed description of the XACML specification is out of the scope of this thesis and can be found in [43].

5.3.3.1 Policy

This section briefly describes the structure of XACML policies. In particular it is focused on the adaptations made in order for the XACML polices to fit to the CDAShip approach. An example structure is given and explained below [11].

```
1 <!ELEMENT Policy (Description?, PolicyDefaults?,
     CombinerParameters?, Target, (CombinerParameters? |
     RuleCombinerParameters? | VariableDefinition | Rule)+,
     Obligations?)>
2 <!ATTLIST Policy
     PolicyId CDATA #REQUIRED
     RuleCombiningAlgId CDATA #REQUIRED>

4 <!ELEMENT Description (#PCDATA)>

6 <!ELEMENT PolicyDefaults (XPathVersion)>
```

[11] *Some low level elements have been omitted at this point in order to provide better readability. Further information can be found in the XACML documentation [43]*

```
 7 <!ELEMENT XPathVersion (#PCDATA)>

 9 <!ELEMENT CombinerParameters (CombinerParameter*)>
10 <!ELEMENT CombinerParameter (AttributeValue)>
11 <!ATTLIST CombinerParameter ParameterName CDATA #REQUIRED>

13 <!ELEMENT RuleCombinerParameters (CombinerParameter*)>
14 <!ATTLIST RuleCombinerParameters    RuleIdRef CDATA #REQUIRED>

16 <!ELEMENT Rule (Description?, Target?, Condition?)>
17 <!ATTLIST Rule  RuleId CDATA #REQUIRED  Effect (Permit | Deny) #
      REQUIRED>

19 <!ELEMENT Target (Subjects?, Resources?, Actions?, Environments?)>

21 <!ELEMENT Subjects (Subject+)>
22 <!ELEMENT Subject (SubjectMatch+)>

24 <!ELEMENT Resources (Resource+)>
25 <!ELEMENT Resource (ResourceMatch+)>

27 <!ELEMENT Actions (Action+)>
28 <!ELEMENT Action (ActionMatch+)>

30 <!ELEMENT Environments (Environment+)>
31 <!ELEMENT Environment (EnvironmentMatch+)>

33 <!ELEMENT SubjectMatch (AttributeValue, (
      SubjectAttributeDesignator | AttributeSelector))>
34 <!ATTLIST SubjectMatch MatchId CDATA #REQUIRED>

36 <!ELEMENT ResourceMatch (AttributeValue,   (
      ResourceAttributeDesignator | AttributeSelector))>
37 <!ATTLIST ResourceMatch MatchId CDATA #REQUIRED>

39 <!ELEMENT ActionMatch (AttributeValue, (ActionAttributeDesignator
      | AttributeSelector))>
40 <!ATTLIST ActionMatch   MatchId CDATA #REQUIRED>

42 <!ELEMENT EnvironmentMatch (AttributeValue, (
      EnvironmentAttributeDesignator | AttributeSelector))>
43 <!ATTLIST EnvironmentMatch  MatchId CDATA #REQUIRED>

45 <!ELEMENT VariableDefinition (ActionAttributeDesignator |
      EnvironmentAttributeDesignator | ResourceAttributeDesignator |
      SubjectAttributeDesignator | VariableReference | Function |
      AttributeValue | Apply | AttributeSelector)>
46 <!ATTLIST VariableDefinition    VariableId CDATA #REQUIRED>

48 <!ELEMENT Expression EMPTY>
```

```
<!ELEMENT VariableReference EMPTY>
<!ATTLIST VariableReference VariableId CDATA #REQUIRED>

<!ELEMENT AttributeSelector EMPTY>
<!ATTLIST AttributeSelector
   RequestContextPath CDATA #REQUIRED
   DataType CDATA #REQUIRED >

<!ELEMENT AttributeValue (#PCDATA)>
<!ATTLIST AttributeValue DataType CDATA #REQUIRED>

<!ELEMENT Function EMPTY>
<!ATTLIST Function  FunctionId CDATA #REQUIRED>

<!ELEMENT Condition (ActionAttributeDesignator |
   EnvironmentAttributeDesignator | ResourceAttributeDesignator |
   SubjectAttributeDesignator | VariableReference | Function |
   AttributeValue | Apply | AttributeSelector)>

<!ELEMENT Apply (ActionAttributeDesignator |
   EnvironmentAttributeDesignator | ResourceAttributeDesignator |
   SubjectAttributeDesignator | VariableReference | Function |
   AttributeValue | Apply | AttributeSelector)*)>
<!ATTLIST Apply FunctionId CDATA #REQUIRED>

<!ELEMENT Obligations (Obligation+)>
<!ELEMENT Obligation (AttributeAssignment*)>
<!ATTLIST Obligation ObligationId CDATA #REQUIRED
   FulfillOn (Permit | Deny) #REQUIRED>

<!ELEMENT AttributeAssignment (#PCDATA)>
<!ATTLIST AttributeAssignment
   DataType CDATA #REQUIRED
   AttributeId CDATA #REQUIRED>
```

<**Policy**> contains a RuleCombiningAlgId, which indicates the algorithm used to combine the different rules of the policy. In our approach, the rule combining algorithm should be deny-overrides, because if the access to a section is denied then the policy should result with a deny and the <Obligations> element should contain the section that has to be encrypted.

<**Description**> contains the description of the policy.

<**PolicyDefaults**> defines a set of default values applicable to the policy. For

example the used xpath version.

<**Target**> defines the applicability of a policy regarding to the request. If all the information in the target matches the information given in the request then the policy is applicable.

<**Subjects**> contains information that have to match to the subject in the request.

<**Resources**> contains information that have to match to the resource in the request. In our approach this would contain xpath queries on the CDA document.

<**Actions**> contains the action that is applied to the policy. In our approach the action has to be "read".

<**Environments**> contains a set of attributes of the environment.

<**Rule**> includes a sequence of rules that have to be combined according to the RuleCombiningAlgId attribute. Rules whose <Target> elements match the request have to be taken into account, others should be ignored.

<**Obligations**> contains obligations that must be fulfilled with the authorization decision. In our approach the obligations contain the information about the sections that have to be encrypted. For further details about the obligations in our approach see 5.3.3.1.

XACML also allows combining several policies in a PolicySet. A detailed description of the PolicySet is out of the scope of this thesis, but can be found in [43]. One important fact of the PolicySet is the policy combining algorithm, which defines how to handle multiple policies that apply to a request. Some of the default XACML policy combining algorithm are deny-overrides, permit-overrides or first-applicable. The disadvantage of these combining algorithms is that the policy evaluation stops as soon as the first policy with the needed result occurs. In our approach it is necessary to evaluate all policies in order to find all sections that have to be secured. Therefore, we defined a new "resolve-all-and-

collect-obligations" policy combining algorithm that resolves all applicable policies and in case a policy evaluation results in a deny, the algorithm collects the policy's obligations. Furthermore, the CDACipher will process the obligations and encrypt the unauthorized sections.

If an error occurred while evaluating a policy that applies to the given request, the policy evaluation result is "Indeterminate". Our combining algorithm will stop and throw an error message.

Obligations

If a policy denies access to a section then the <Obligation> element defines which section has to be encrypted in order to allow the document exchange. According to the XACML schema definition each <Obligation> element has to contain an ObligationID attribute. We assign the value "Encrypt" to the ObligationID attribute to indicate the action to be taken. Further, each obligation has to have two <AttributeAssignment> elements. One with the attribute AttributeID with value "Message", which gives free-form description of the obligation and the section that has to be encrypted. The second <AttributeAssignment> has the attribute AttributeID with value "pathMessage", which contains the xpath to the section that has to be encrypted.

A sample obligation element is shown below. In this example the document is only allowed to be exchanged, if it does not contain any address information.

```
1 <Obligations>
2   <Obligation ObligationId="Encrypt" FulfillOn="Deny">
3     <AttributeAssignment AttributeId="Message" DataType="
        http://www.w3.org/2001/XMLSchema#string">
4         In order to send this CDA document the address
            field needs to be encrypted.
5     </AttributeAssignment>
6     <AttributeAssignment AttributeId="pathMessage"
        DataType="http://www.w3.org/2001/XMLSchema#string">
7         //md:ClinicalDocument/md:recordTarget/md:
            patient/md:addr/md:streetAddressLine
8     </AttributeAssignment>
```

```
 9    </Obligation>
10  </Obligations>
```

5.3.3.2 Request

The CDAShip automatically generates a request for the given CDA document. In order to create the request, the CDA document as well as information about the recipient have to be provided. The information about the recipient is needed to construct the subject element of the request. The structure of the request is given and explained below.

```
 1  <!ELEMENT Request (Subject+, Resource+, Action, Environment)>

 3  <!ELEMENT Subject (Attribute*)>
 4  <!ATTLIST Subject SubjectCategory CDATA #REQUIRED>

 6  <!ELEMENT Resource (ResourceContent?, Attribute*)>
 7  <!ELEMENT ResourceContent (#PCDATA)>

 9  <!ELEMENT Action (Attribute*)>

11  <!ELEMENT Environment (Attribute*)>

13  <!ELEMENT Attribute (AttributeValue+)>
14  <!ATTLIST Attribute

        AttributeId CDATA #REQUIRED
        DataType CDATA #REQUIRED
        Issuer CDATA #IMPLIED>

16  <!ELEMENT AttributeValue (#PCDATA)>
```

<**Subject**> specifies a subject by listing a sequence of <Attribute> elements associated with the subject. In our approach the subject is the receiver of the CDA document.

<**Resource**> includes information about the CDA document to which access is requested by listing a sequence of <Attribute> elements associated with the resource. In our approach the generated request includes the <ResourceContent> element, which contains the whole requested CDA

document[12]. This is necessary in order to allow the use of xpath queries when writing the policies.

<**Action**> specifies the requested action on the resource. In our case the action would be read.

<**Environment**> contains a set of attributes of the environment. Environment attributes are not associated with the resource, the action or any of the subjects of the access request. Therefore, in our approach the <Environment> element does not include any content.

5.3.3.3 Response

The policy evaluation creates a response document, which is processed by the CDAChiper component. For our project processing the obligations is the most important part of the response, since they specify which sections have to be encrypted.

The structure of the response is as follows[13].

```
1  <!ELEMENT Response (Result+)>

3  <!ELEMENT Result (Decision, Status?, xacml:Obligations?)>
4  <!ATTLIST Result ResourceId CDATA #IMPLIED>

6  <!ELEMENT Decision (#PCDATA)>

8  <!ELEMENT Status (StatusCode, StatusMessage?, StatusDetail?)>

10 <!ELEMENT StatusCode (StatusCode?)>
11 <!ATTLIST StatusCode    Value CDATA #REQUIRED>

13 <!ELEMENT StatusMessage (#PCDATA)>

15 <!ELEMENT StatusDetail (#PCDATA)>
```

<**Decision**> contains the decision of the policy evaluation. The decision may be "Permit", "Deny", "Indeterminate" or "NotApplicable".

[12]The included document now has the namespace prefix "md". This has to be taken into account when writing the xpath statements in the policies.

[13] *where xacml:Obligations refers to the Obligation element of the XACML Schema presented in Section 5.3.3.1*

<**Status**> indicates whether errors occurred during evaluation of the request.

<**Obligation**> includes the obligations from the policies or policy set that were applicable to the given request. For details of the <Obligation> element see 5.3.3.1.

Chapter 6

Evaluation

This chapter describes the evaluation of the introduced CDAShip and shows that the described concept meets the hypothesis stated in Chapter 2.

First a description of a sample application, which implements the abstract components of the CDAShip is given. This is followed by an introduction of the sample CDA documents and XACML policies used for our example scenario. It is important to note that both, the sample application as well as the scenario, use information from the e-MS (electronic Medical Summary) project from the Vancouver Island Health Authority. The e-MS is a subset of patient data suitable for communication amongst Primary Health Care (PHC) providers for the purpose of sharing the care of an individual patient. The e-MS project has developed a core data set for the following types of CDA documents: referral letter, consulting report and medical summary. A detailed description of the core data set can be found at [2].

The introduction of the sample document is followed by an example scenario demonstrating the use of the CDAShip and a discussion of the limitations of the CDAShip.

6.1 Sample Application

In order to test and evaluate the described CDAShip, the abstract components Transport, FileSystem and Audit have also been implemented. The implementation of these components is fairly simple and only used as proof of concept to demonstrate the use and

functionality of the CDAShip. When using the CDAShip in a real world application these components, should be rigorously developed, and made more secure.

The following sections describe the sample implementation of the abstract components.

6.1.1 Design of Abstract Components

6.1.1.1 Transport

As Transport component a simple email client has been created. The email client is a sample implementation of Sun's JavaMail API [30] and does not have a user interface, since the exchange of the documents should not be directly visible to the caregiver.

When initializing the Transport component the following information has to be provided

- the user's email address
- the user name
- the password for the email account
- the smtp host
- the imap host
- the location of the public key
- the type of the public key
- the name of the folder, which should be checked for new messages
- the frequency, indicating how often the emails should be checked.

Using the given information, the Transport component creates a connection to the imap-server and regularly checks for new emails. It also enables the sending of emails.

Whenever a CDA document is sent the subject line of the email is *[CDAShip] CDA Document* and the email includes the CDA document, the signature key information document, all applied policies as well as the public key information of the sender as attachment.

An email that contains a key share has the subject line *[CDAShip] KeyShare: KeyShare_[secret_key_id]* and includes the key share, the signature key information document and the sender's public key information.

In case the Transport component exchanges a key share request, the subject line is *[CDAShip] KeyShareRequest: ID=[secret_key_is]* and the email has the request, the signature key information document as well as the sender's public key information attached.

Once a key share request has been received and processed the requested key share will be exchanged in an email with the subject line *[CDAShip] Requested KeyShare: KeyShare_[secret_key_id]* and includes the same documents as a regular key share exchange.

The last type of email that the Transport component exchanges is the notification email to the patient, which includes the request document and has the subject line *[CDAShip-Warning] Unauthorized data has been accessed.*

The Transport component will check for new emails based on the frequency value defined at component initialization. When a new mail arrives the Transport component reads the subject line and collects the attachments. Depending on the subject line it throws the appropriate event to notify the Import/Export component about the arrival of the email. The Import/Export component then triggers the processing of the information.

It is important to note that this is only a primitive email client used to show the use of the CDAShip. In a real world example the Transport component has to implement more security features such as better authentication. Also it is more secure not to include the ids of the documents in the subject line, since the subject line may be exposed to others who do not have the right to see the information in the message bodies. A different approach may be to include the ids in the message bodies. In addition, other features may be implemented to make the email client more stable such as the recreation of the connection to the email account after time out.

6.1.1.2 FileSystem

The sample implementation of the FileSystem component takes a given file and saves it in a specified folder on the hard drive. Again this is only used to prove the concept of the CDAShip. For practical use security layers have to be implemented. This would include the protection of the saved file through either encrypting it before saving or specifying special access rights and password protection, in order to ensure that no other person than the actual recipient is able to access the received document.

The folder is set when initializing the FileSystem component. The component creates three subfolders, one for received CDA documents, one for received key share documents and another one for decrypted CDA documents in case an encrypted section has been decrypted.

In order to be able to retrieve CDA and key share documents at a later time, the FileSystem includes the document ID in the file name before saving. CDA documents will have a file name of the following form: *CDA_extension_[extension_number]_root_[root_number]*, where the extension number as well as the root number are specified in the ID element of the CDA document. The key share documents will have a name of this form: *KeyShare_[secret_key_id]*, with the secret key id given in the key share document.

6.1.1.3 Audit

The Audit component consists of a small mySQL database that stores information about the key shares that have been distributed upon requests. Basically, the database has two tables, one includes the information about the shareholders and the other includes the request data. The schema of the database is shown in Figure 6.1

It is important to note that the fields "requester" and "forward" in the request table are foreign keys from the shareholder table and that the "forward" field only contains information if the reason for the request is the forwarding of a document. All other information is self-explanatory and taken directly from the request document that was passed to the audit

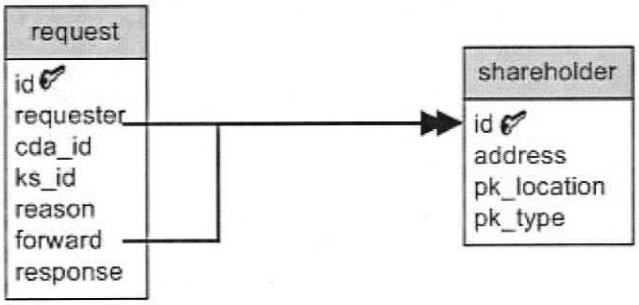

Figure 6.1. *Audit database schema*

component.

Similar to the two other abstract components the Audit component is only designed for proof of concept and enhanced security layers are missing. For example, the database should be stored on a secure place and the access to it should also be controlled.

6.1.1.4 User Interface

In addition to the three abstract components that have been described above, a test application was developed as well. It is used to first of all gather the information needed to initialize the abstract components as well as the CDAShip itself. When starting the test application it will prompt amongst others for the email settings, for the directory used by the FileSystem to store and retrieve the documents, and for the settings used by the KeyManager to determine the amount of key shares as well as the threshold value. All these settings are saved to a file and loaded again, to avoid the user typing the same information every time the program is started.

After all the settings have been applied and the components as well as the CDAShip have been initialized, the program gives the user the option to either send a CDA document or to access a section of a given document. Both scenarios are briefly described below.

Sending a CDA document

When the "Send CDA" button is pressed, the application opens a new win-

dow requesting information about the recipient of the CDA document. This information is needed in order to create the subject of the XACML request.
After the information about the recipient has been entered and confirmed the application opens another form requesting further information including the location of the CDA document, the location of the policy that has to be applied, to whom the document should be sent and which shareholders should receive a key share in case the document will be encrypted. The form also includes a check box indicating whether the document should be forwarded. This check box has to be checked, if a CDA document should be forwarded to another caregiver. In this case the patient's email as well as the reason for forwarding has to be entered. If this option is checked, the application calls the forwardDocument method from the Import/Export component and not the sendCDA method once the send button is pressed.
A screenshot of this form is shown in Figure 6.2

Accessing an unauthorized section

In case the "Access Section" button is pressed the application also opens a form requesting information about the recipient needed for the subject section in the XACML request. Once the information has been confirmed the application opens the form shown in Figure 6.3. This form requests the location of the CDA document, which contains the encrypted section and the location of the policy, which has to be applied to the document. Once the CDA document has been specified, the application parses the CDA document and displays all document nodes. If a node is encrypted, the node will be shown with a lock in front of it. The user has to select which section should be decrypted. It is also necessary to specify the patient's email address and the reason why the section has to be accessed. Again, this application is only used to prove the concept of the CDAShip. In a real world application the patient's email address should be

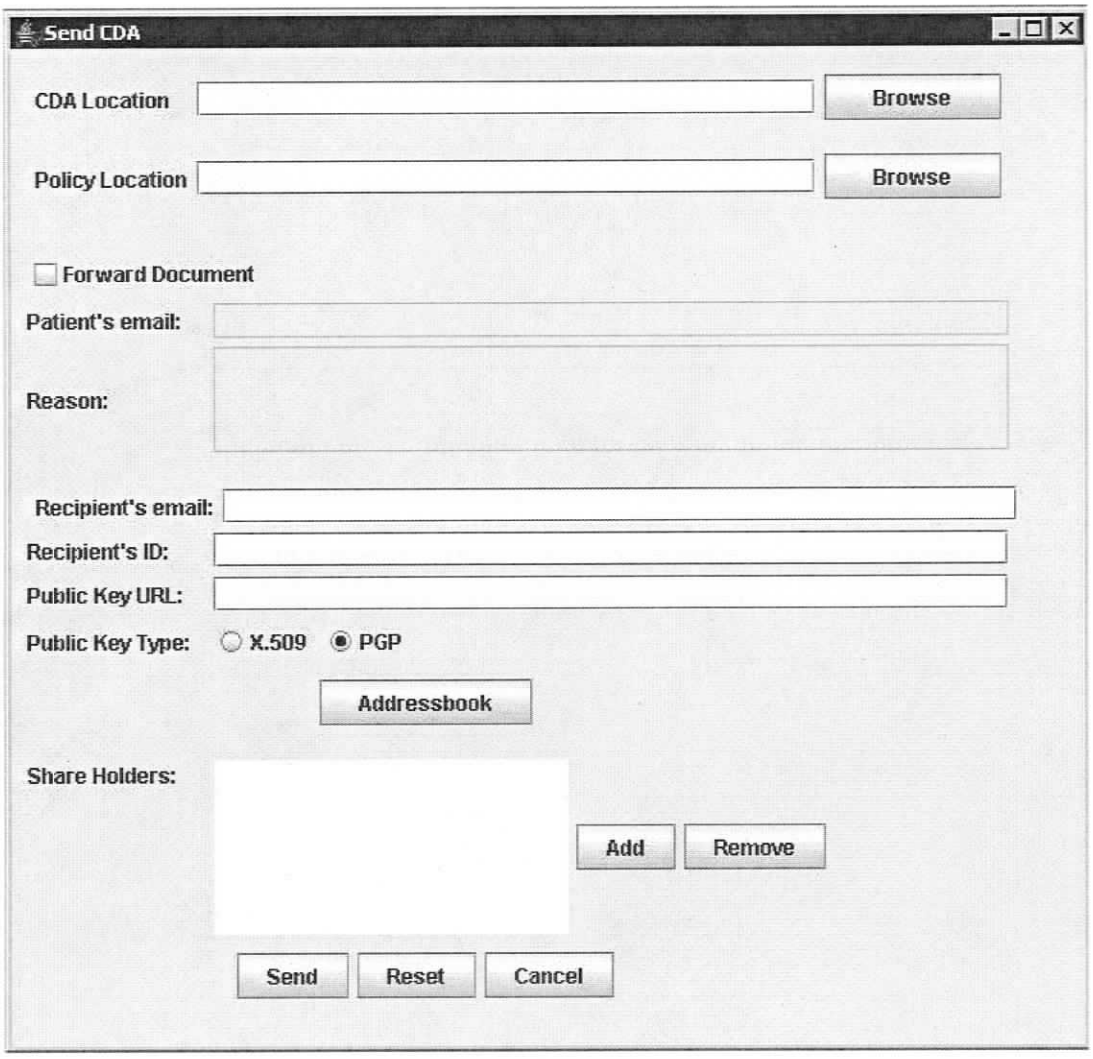

Figure 6.2. *Screenshot: Send CDA*

retrieved automatically from the system in order to avoid the caregiver entering the wrong email address. Once all information has been gathered and the "Send Request" button is pressed the application calls the accessSection method from the Import/Export component.

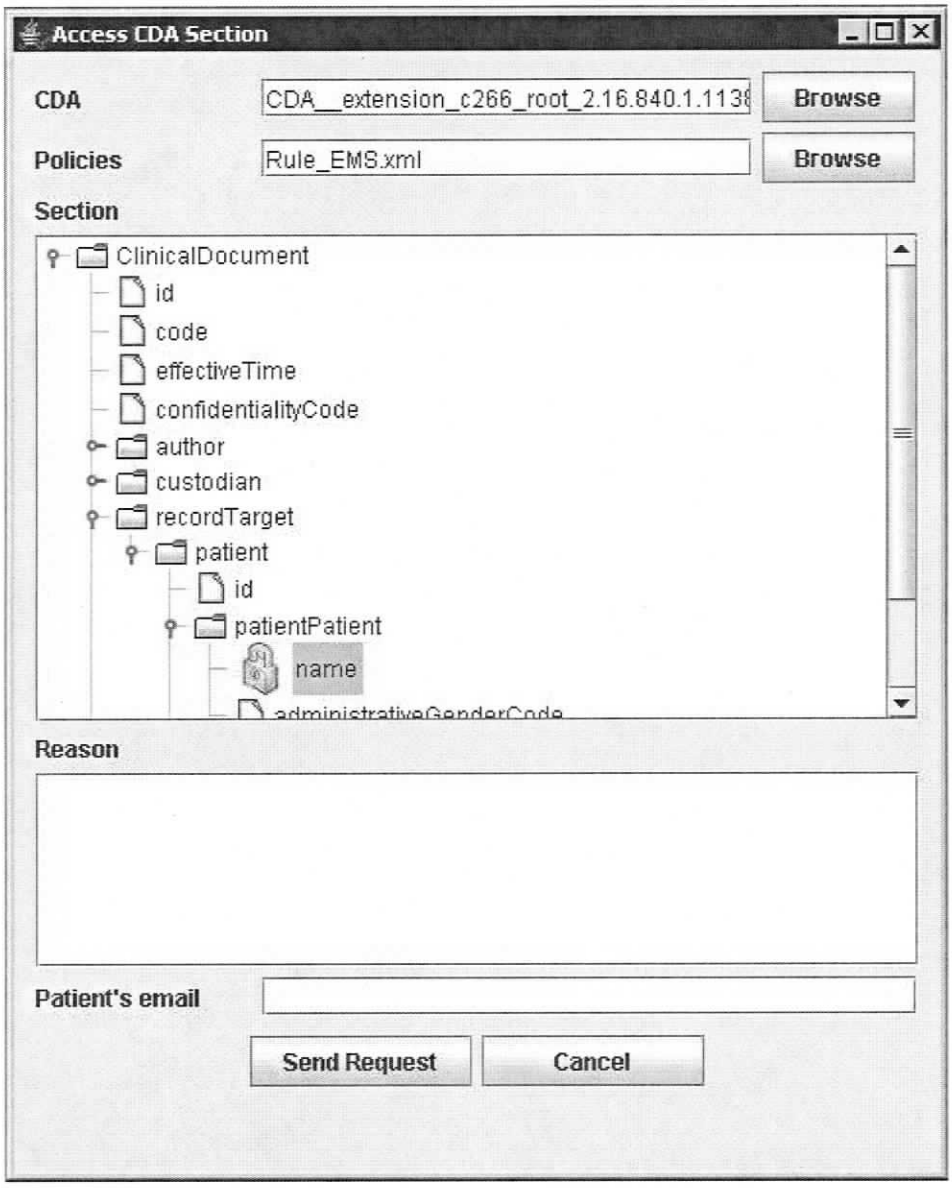

Figure 6.3. *Screenshot: Access Section*

6.2 Sample Documents

This Section describes the sample CDA documents and the sample XACML policies used for the evaluation of the CDAShip.

6.2.1 Sample CDA document

The CDA document used for our testing purpose is a modified e-MS example document. The sample e-MS documents are part of the e-MS standard, which can be downloaded from `http://www.e-ms.ca/documentation.php`. Our e-MS CDA document contains information about the patient, the purpose, allergies as well as the family history. The content of the document is illustrated in Figure 6.4.

Patient: Eve Everywoman , Jr. **MRN:** 9999999999
Consultant: **Created On:** August 9, 2004

Referral Purpose

Please see this patient regarding occasional occurrences of arrhythmia at rest accompanied with mild shortness of breath. Mild exercise leads to faintness and mild chest pain.
Urgency: expedite

Other Provider	Reason
Dr. Vera V. Valve at Level 7 Healthcare, Inc	High blood pressure in the past
Dr. Sara S. Specialize at Good Health Hospital	24 Hour ECG recording

Comments: The patient expressed that this problem has gradually been getting worse over the course of the past year and a half.

Alerts

Alert	Comments	Date
History of violence	Mostly towards family members	1999
History of mental illness	Mild anxiety disorder	August 2000

Allergies

Allergy Type	Severity	Comments	Date
Penicillin	Mild	Penicillin. Resulting in hives. Patient hasn't been tested recently to see if this is still an active allergy	1965
Dairy products	Moderate	Dairy. Resulting in puffy eyes	1970-1985

Family History

The patient's father died of a myocardial infarction at the age of 65.

Signed by: on ,

Figure 6.4. *Sample Referral*

In addition to the data shown in Figure 6.4 the document also contains further meta-data encoded in the xml document, for example the information about the recipient. This meta-data is not necessarily important for the caregiver but rather for the system processing the document. For our scenarios and policies the name of receiving organization as well as

the priority of the referral are important. A snippet of the e-MS CDA document containing this data is shown below. The whole document can be found in Appendix F.

```
...
 <!-- ***** Information Recipient ************************-->
 <informationRecipient typeCode="PRCP">
  <intendedRecipient classCode="ASSIGNED">
   <id extension="ggottschalk"
       root="DCCD2C68-389B-44c4-AD99-B8FB2DAD1234"/>
   <informationRecipient>
    <name>
     <prefix>Dr.</prefix>
     <given>Gudrun</given>
     <family>Gottschalk</family>
    </name>
   </informationRecipient>
   <receivedOrganization>
    <name>Vancouver General Hospital</name>
   </receivedOrganization>
  </intendedRecipient>
 </informationRecipient>
...
 <!-- ***** Purpose ************************-->
   <component>
    <section>
     <code code="001"
        codeSystem="7BA9BFFD-D25F-44e8-A7B0-0DF214D6845B"
        codeSystemName="e-MS Document Section Codes"
        displayName="Purpose"/>
     <title>Referral Purpose</title>
     <text>
      ...
     </text>
     <entry typeCode="DRIV">
      <observation classCode="OBS" moodCode="EVN">
       <code nullFlavor="NA"/>
       <text>Please see this patient regarding occasional occurrences
           of arrhythmia at rest accompanied with mild shortness of
          breath. Mild exercise leads to faintness and mild chest
          pain. </text>
       <priorityCode code="E"
          codeSystem="1261620F-C09D-4825-96E4-E808DCFA4D17"
          codeSystemName="eMS_Purpose_Urgency"
          displayName="Expedite (call)"/>
      </observation>
     </entry>
    </section>
   </component>
```

6.2.2 Sample Policies

For testing purpose two XACML policies and one XACML policy set were created. The policy set combines both policies to one single policy using the resolve-all-collect-obligations policy combining algorithm, which was described in 5.3.3.1. The actual policy set can be found in Appendix I. The two policies, "Receiving Organization" and "Urgent", are introduced in the next two subsections.

6.2.2.1 Policy "Receiving Organization"

The policy "receiving Organization" states:

If the receiving organization is not the Victoria General Hospital, then the e-MS referral should include only the minimal required patient information. All other information has to be encrypted.

In other words this states that in the e-MS CDA, section recordTarget, which includes patient data such as name and address, has to be encrypted such that it is still valid according to the e-MS CDA schema, but does not contain any further information. This requires that all sub-elements other than the id have to be encrypted. Also this policy implies that all e-MS component sections other than the Purpose section have to be encrypted as well, since they include information such as the patient history.

The complete XACML policy that represents the description above can be found in Appendix G. The snippet below shows only the condition and obligations sections of the policy.

```
1  ...
2  <!-- ********* Condition************************************-->
3    <Condition>
4    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
5     <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
         equal">
6      <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">Victoria
          General Hospital</AttributeValue>
7      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
          one-and-only">
```

```
8       <AttributeSelector
           RequestContextPath="//xacml-context:Resource/xacml-context:
           ResourceContent/md:ClinicalDocument/md:informationRecipient/
           md:intendedRecipient/md:receivedOrganization/md:name/text()"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>
9       </Apply>
10     </Apply>
11    </Apply>
12    </Condition>
13   </Rule>
14  <!-- ********* Obligations ********************************-->
15   <Obligations>
16    <Obligation ObligationId="Encrypt" FulfillOn="Deny">
17     <AttributeAssignment
          AttributeId="message"
          DataType="http://www.w3.org/2001/XMLSchema#string">Encrypt
          patient address in the recordTarget.</AttributeAssignment>
18     <AttributeAssignment
          AttributeId="path"
          DataType="http://www.w3.org/2001/XMLSchema#string">/:
          ClinicalDocument/:recordTarget/:patient/:addr</
          AttributeAssignment>
19    </Obligation>
20    <Obligation ObligationId="Encrypt" FulfillOn="Deny">
21     <AttributeAssignment
          AttributeId="message"
          DataType="http://www.w3.org/2001/XMLSchema#string">Encrypt
          patient phone number.</AttributeAssignment>
22     <AttributeAssignment
          AttributeId="path"
          DataType="http://www.w3.org/2001/XMLSchema#string">/:
          ClinicalDocument/:recordTarget/:patient/:telecom</
          AttributeAssignment>
23    </Obligation>
24    <Obligation ObligationId="Encrypt" FulfillOn="Deny">
25     <AttributeAssignment
          AttributeId="message"
          DataType="http://www.w3.org/2001/XMLSchema#string">Encrypt
          patientPatient.</AttributeAssignment>
26     <AttributeAssignment
          AttributeId="path"
          DataType="http://www.w3.org/2001/XMLSchema#string">/:
          ClinicalDocument/:recordTarget/:patient/:patientpatient</
          AttributeAssignment>
27    </Obligation>
28    <Obligation ObligationId="Encrypt" FulfillOn="Deny">
29     <AttributeAssignment
          AttributeId="message"
          DataType="http://www.w3.org/2001/XMLSchema#string">Encrypt all
          sections, except of the Purpose section (e-MS code =
```

```
30  001).</AttributeAssignment>
31      <AttributeAssignment
            AttributeId="path"
            DataType="http://www.w3.org/2001/XMLSchema#string">/:
            ClinicalDocument/:component/:structuredBody/:component/:section
            /:code[@code!=001]/parent::*</AttributeAssignment>
32    </Obligation>
33   </Obligations>
34  </Policy>
```

The condition statement is used to determine whether the rule of the policy applies to given request. In our case the rule applies if the receiving organization is not the Victoria General Hospital. Lines 3 to 14 define the condition statement using nested apply functions. The *string-one-and-only* function from lines 9 to 11 retrieves the value of the receiving organization from the request document by using an xpath expression. It also makes sure there is only one result of the xpath expression. Further, the *string-equal* function (lines 5 to 12) compares the xpath result with the string "Victoria General Hospital" and evaluates to true if they are the same. Since the policy states that encryption has to be done if the document is sent to an organization other than the Victoria General Hospital, the result of the *string-equal* function has to be reversed using the *not* function. This is done in lines 4 to 13.

Furthermore, the obligations are the most important part of the policy, since they define what has to be encrypted. The obligations are described in lines 17 to 49. This policy contains 4 obligations. Each obligation consists of two attributes as described in Section 5.3.3.1. The first three obligations (lines 18 to 40) specify that the elements addr, telecom and patientpatient from the recordTarget element have to be encrypted. And the last obligation, from lines 41 to 48, specifies that all sections except the purpose section have to be encrypted. The purpose section is defined with the e-MS code 001.

6.2.2.2 Policy "Urgent"

The second policy used to test the CDAShip is called "Urgent" and can be summarized as:

If e-MS document has to be exchanged, but the purpose of the exchange is not urgent

then only the purpose section and the family history section should be exchanged, all other component sections have to be encrypted.

The priority of the document is stored in the purpose section in the priorityCode element. Each priorityCode element contains a code attribute, which specifies the priority level. The e-MS project distinguishes between three levels of priority: Expedite (code="E"), Urgent (code="U") and Routine (code="R"). This means that each document, which does not have a priorityCode of U has to encrypt all component section except the purpose and family history sections.

Since the structure of the condition and obligation sections of the policy are similar to those of the "Receiving Organization" policy (see 6.2.2.1), they will not be further discussed here. However, the policy that defines this rule can be found in Appendix H.

6.3 Sample Scenario

In our example scenario, we use a (2,3)-threshold scheme, which means that 2 out of 3 key shares are required in order to reconstruct the secret key. Further, we use the Blowfish encryption algorithm for secret key encryption and the RSA algorithm for public key encryption.

We exchange the sample e-MS CDA document, described in Section 6.2.1, between the following three caregivers: Dr. Patrick Pump and Dr. Joe Frozen, who are both members of the Victoria General Hospital and Dr. Gudrun Gottschalk, who is a member of the Royal Victoria Hospital. All three caregivers are also shareholders in our example.

Dr. Patrick Pump refers the patient to Dr. Gudrun Gottschalk. The referral is not urgent, which means that the policy "Urgent" will apply to this referral and since Dr. G. Gottschalk is not a member of the Victoria General Hospital the policy "Receiving Organization" will also apply. With both policies applying, the received document will not contain information such as the allergies or family history. It will also not contain the patient's name or any other personal information in the recordTarget section. The XML code below shows a snippet of

the document that Dr. G. Gottschalk received. It is important to note that all information that has not been changed from the original document and repeating information has been replaced with "..." for conciseness and to draw attention to the sections that have been modified by the CDAShip.

```
...
<!-- ***** Information Recipient ************************-->
 <informationRecipient typeCode="PRCP">
  <intendedRecipient classCode="ASSIGNED">
   <id extension="ggottschalk"
       root="DCCD2C68-389B-44c4-AD99-B8FB2DAD1234"/>
   <informationRecipient>
    <name>
     <prefix>Dr.</prefix>
     <given>Gudrun</given>
     <family>Gottschalk</family>
    </name>
   </informationRecipient>
   <receivedOrganization>
    <name>Vancouver General Hospital</name>
   </receivedOrganization>
  </intendedRecipient>
 </informationRecipient>
...
 <!-- ***** Record Target ************************-->
 <recordTarget contextControlCode="OP" typeCode="RCT">
  <patient classCode="PAT">
   <id assigningAuthorityName="BC-PHN" extension="9999999999"
       root="2BFBA1E9-79C2-4bbb-B589-41B949BD6A3B"/>
   <id assigningAuthorityName="WCB" extension="99999-9"/>
   <addr>
    <xenc:EncryptedData>
     <xenc:EncryptionMethod Algorithm="Blowfish"/>
     <ds:KeyInfo>
      <ds:KeyName>fP8NVF7TIKWlDZm1fobfUA==</ds:KeyName>
     </ds:KeyInfo>
     <xenc:CipherData>
      <xenc:CipherValue>v/nlNoX3J0y0KEmCXnH...</xenc:CipherValue>
     </xenc:CipherData>
    </xenc:EncryptedData>
   </addr>
   <patientPatient>
    <xenc:EncryptedData>
     ...
    </xenc:EncryptedData>
   </patientPatient>
  </patient>
 </recordTarget>
```

```
...
<!-- ***** e-MS CDA Structured Body ******************-->
 <component>
  <structuredBody>
   <!-- ***** Purpose *************************-->
   <component>
    <section>
     <code code="001"
        codeSystem="7BA9BFFD-D25F-44e8-A7B0-0DF214D6845B"
        codeSystemName="e-MS Document Section Codes"
        displayName="Purpose"/>
     <title>Referral Purpose</title>
     <text>
      ...
     </text>
     <entry typeCode="DRIV">
      <observation classCode="OBS" moodCode="EVN">
       <code nullFlavor="NA"/>
       <text>
        ...
       </text>
       <priorityCode code="E"
          codeSystem="1261620F-C09D-4825-96E4-E808DCFA4D17"
          codeSystemName="eMS_Purpose_Urgency"
          displayName="Expedite (call)"/>
      </observation>
     </entry>
    </section>
   </component>
   <!-- ***** Allergies *************************-->
   <component>
    <section>
     <xenc:EncryptedData>
       ...
     </xenc:EncryptedData>
    </section>
   </component>
   <!-- ***** Family History *************************-->
   <component>
    <section>
     <xenc:EncryptedData>
      ...
     </xenc:EncryptedData>
    </section>
   </component>
  </structuredBody>
 </component>
</ClinicalDocument>
```

The code snippet above clearly shows that the sections that had to be encrypted are

replaced by the <xenc:EncryptedData> element, which contains the section data in encrypted form as well as information such as encryption algorithm and name of the used secret key.

In addition to the e-MS CDA document, Dr. G. Gottschalk has also received a key share document including an encrypted share of the secret key used to encrypt the sections of the e-MS CDA document as well as information about the other shareholders. Both of the other caregivers also received a key share document containing their share of the secret key. Dr. G. Gottschalk's key share document[1] is shown below. The structure of a key share document was discussed in 5.3.2.1.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <KeyShare xmlns="http://www.ppci.ca/CDAShip/KeyShare#"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.ppci.ca/CDAShip/KeyShare# http://
       www.cs.uvic.ca/%7Eobry/CDAShip/Schema/KeyShare.xsd"
       Id="fP8NVF7TIKWlDZm1fobfUA==" >
3    <xenc:EncryptedKey>
4     <xenc:EncryptionMethod Algorithm="RSA"/>
5     <ds:KeyInfo>
6      <ds:KeyName>ggotschalk_public_key</ds:KeyName>
7     </ds:KeyInfo>
8     <xenc:CipherData>
9      <xenc:CipherValue>P92H/NycsmKVAIX...</xenc:CipherValue>
10    </xenc:CipherData>
11    <xenc:CarriedKeyName>fP8NVF7TIKWlDZm1fobfUA==</xenc:CarriedKeyName>
12   </xenc:EncryptedKey>
13   <KeyName>fP8NVF7TIKWlDZm1fobfUA==</KeyName>
14   <SecretKeyEncryptionMethod>Blowfish</SecretKeyEncryptionMethod>
15   <KeyShareInfo>
16    <ShareNumber>1</ShareNumber>
17    <Threshold>2</Threshold>
18    <TotalShares>3</TotalShares>
19    <Prime>557535347994796531677916496955397621 67</Prime>
20    <NegativeSecret>false</NegativeSecret>
21    <ExpiryDate>
22     <Month>2</Month>
23     <Day>1</Day>
24     <Year>2007</Year>
25    </ExpiryDate>
26    <ShareHolders>
```

[1]The actual cipher value has been shortened.

```
27      <ShareHolder>
28       <Address>ppump@med.ca</Address>
29       <ID>ppump</ID>
30       <PublicKeyInfo>
31        <PublicKeyLocation>http://www.cs.uvic.ca/~obry/CDAShip/ppump.asc
             </PublicKeyLocation>
32        <PublicKeyType>PGP</PublicKeyType>
33       </PublicKeyInfo>
34      </ShareHolder>
35      <ShareHolder>
36       <Address>ggottschalk@med.ca</Address>
37       <ID>ggottschalk</ID>
38       <PublicKeyInfo>
39        <PublicKeyLocation>http://www.cs.uvic.ca/~obry/CDAShip/
             ggottschalk.asc</PublicKeyLocation>
40        <PublicKeyType>PGP</PublicKeyType>
41       </PublicKeyInfo>
42      </ShareHolder>
43      <ShareHolder>
44       <Address>jfrozen@med.ca</Address>
45       <ID>jfrozen</ID>
46       <PublicKeyInfo>
47        <PublicKeyLocation>http://www.cs.uvic.ca/~obry/CDAShip/jfrozen.
             asc</PublicKeyLocation>
48        <PublicKeyType>PGP</PublicKeyType>
49       </PublicKeyInfo>
50      </ShareHolder>
51     </ShareHolders>
52    </KeyShareInfo>
53   </KeyShare>
```

As next step Dr. Gudrun Gottschalk needs to forward the document to Dr. Joe Frozen, because she will not be in town for the next 2 weeks and therewith can not treat the patient. The only changes Gudrun Gottschalk made to the document are in the section informationRecipient. This section now contains the information about the new recipient, Dr. J. Frozen. The CDAShip will then automatically request the key shares for the document from all shareholders and apply the policies again. Dr. Joe Frozen, is a member of the Victoria General Hospital, therefore he should get access to the patient history section as well as the patient address and name. The only information that should be encrypted is the name of the patient, since the referral is still not urgent. A snippet of the document that arrived at Dr. Joe Frozen's CDAShip is shown below. It shows that the family history section as well as the recortTarget have been decrypted.

```
...
<!-- ***** Information Recipient ************************-->
<informationRecipient typeCode="PRCP">
 <intendedRecipient classCode="ASSIGNED">
  <id extension="jfrozen"
     root="DCCD2C68-389B-44c4-AD99-B8FB2DAD5678"/>
  <informationRecipient>
   <name>
    <prefix>Dr.</prefix>
    <given>Joe</given>
    <family>Frozen</family>
    <suffix>Sr.</suffix>
   </name>
  </informationRecipient>
  <receivedOrganization>
   <name>Victoria General Hospital</name>
  </receivedOrganization>
 </intendedRecipient>
</informationRecipient>
...
<!-- ***** Record Target ************************-->
<recordTarget contextControlCode="OP" typeCode="RCT">
 <patient classCode="PAT">
  <id assigningAuthorityName="BC-PHN" extension="9999999999"
     root="2BFBA1E9-79C2-4bbb-B589-41B949BD6A3B"/>
  <id assigningAuthorityName="WCB" extension="99999-9"/>
  <addr><streetAddressLine>2222 Home Street</streetAddressLine>
   <city>Ann Arbor</city>
   <state>BC</state>
   <postalCode>A1B 2C3</postalCode>
   <country>CA</country></addr>
  <patientPatient>
   <name>
    <prefix>Mrs.</prefix>
    <given>Eve</given>
    <family>Everywoman</family>
   </name>
  </patientPatient>
 </patient>
</recordTarget>
...
<!-- ***** e-MS CDA Structured Body ************************-->
<component>
 <structuredBody>
  <!-- ***** Purpose ************************-->
  <component>
   <section>
    <code code="001"
       codeSystem="7BA9BFFD-D25F-44e8-A7B0-0DF214D6845B"
       codeSystemName="e-MS Document Section Codes"
```

```
          displayName="Purpose"/>
47       <title>Referral Purpose</title>
48       ...
49       </entry>
50      </section>
51     </component>
52     <!-- ***** Allergies ************************-->
53     <component>
54      <section>
55       <xenc:EncryptedData>
56        <xenc:EncryptionMethod Algorithm="Blowfish"/>
57        <ds:KeyInfo>
58         <ds:KeyName>GgMflTxjOZeKvTE57wKa7w==</ds:KeyName>
59        </ds:KeyInfo>
60        <xenc:CipherData>
61         <xenc:CipherValue>B8ku++bsE+uidrMOgo</xenc:CipherValue>
62        </xenc:CipherData>
63       </xenc:EncryptedData>
64      </section>
65     </component>
66     <!-- ***** Family History ************************-->
67     <component>
68      <section>
69       <code code="10157"
           codeSystem="2.16.840.1.113883.6.1"
           codeSystemName="LOINC"
           displayName="Family History"/>
70       ...
71      </section>
72     </component>
73    </structuredBody>
74   </component>
```

Assuming that Dr. Joe Frozen now needs to access the allergy section of the document, since he believes that the patient may be allergic to the medication he wants to prescribe. Dr. Joe Frozen requests the decryption of the allergy section. Further, it is assumed that Dr. Gudrun Gottschalk is currently not available and has shut down her CDAShip, which means that she is not able to provide her key share. This means that in order to decrypt, only the key shares from Dr. Joe Frozen and Dr. Patrick Pump are available, which is enough to reconstruct the secret key. Using the two received key shares Dr. Joe Fronzen's CDAShip is able to decrypt the section and retrieve a document that contains all the information as shown in Figure 6.4.

Meanwhile the patient has received two emails, informing her about the access to the

sections. One access was made to forward the document to Dr. Joe Frozen and the other privacy violation was in order to retrieve the patient's allergies. If the patient does not agree that these violations were in her interest then she can take further legal steps.

Also, both of the violations have been audited in the database from Dr. Patrick Pump. The entries of Dr. P. Pump's audit trails are shown in Table 6.1 and 6.2. In Table 6.2 the response column was omitted, since its content is very large. However, it shows that the requests made were both for the same e-MS CDA document, since the CDA_id is the same, but that the exchanged key shares were different. Also it shows the reason for the request and that the first request was used in order to forward the document to Dr. J. Frozen. It is important to mention that Dr. G. Gottschalk's audit trail database differs from this one, because it only has one entry in the request database, since she did not deliver the second secret key.

id	**address**	**pk_location**	**pk_type**
jfrozen	jfrozen@med.ca	http://www.cs.uvic.ca/~obry/CDAShip/jfrozen.asc	PGP
ggottschalk	ggottschalk@med.de	http://www.cs.uvic.ca/~obry/CDAShip/ggottschalk.asc	PGP

Table 6.1. *Database table shareholder*

id	**requester**	**cda_id**	**ks_id**	**reason**	**forward**
1	ggottschalk	_extension_DEAF84EC-87F7-4bff-BDB6-9DF00644D80D_root_2.1.6840.1.113883.3.933_	iSyE5y+KrylugX01Mt4tEw==	[CDAShip Forward] refer to Dr. J. Frozen	jfrozen
2	jfrozen	_extension_DEAF84EC-87F7-4bff-BDB6-9DF00644D80D_root_2.1.6840.1.113883.3.933_	GgMflTxjOZeKvTE57wKa7w==	Need to see allergies	NULL

Table 6.2. *Database table request*

This small sample scenario has shown that it is possible to exchange patient data in

a more private way by encrypting certain document sections according to specified policies. Further, it also showed that different caregivers may be given different information according to the policies. Especially, it is important to mention that a caregiver can receive information from other caregivers who themselves did not have access to the information. For example, Dr. Gudrun Gottschalk did not have access to the allergy and family history whereas Dr. Joe Frozen, who received information from Dr. Gudrun Gottschalk was able to access this information.

This evaluation has shown that it possible to exchange highly sensitive data among collaborating caregivers while ensuring privacy and confidentiality and providing the flexibility to act in emergency cases. In detail the introduced CDAShip is able

- restricts access to particular sensitive information based on patient and clinician's consent as well as legislative requirements
- to ensure that such restrictions can be overridden on a need-to-know basis only
- to monitor, audit and report the overriding of privacy policies.

6.4 Limitations

The implementation of the CDAShip is still in the initial stage and therefore has some limitations.

Some of the constraints of the CDAShip have already been discussed in Section 5.1. These limitations are bound to the decision to use Java as implementation language. The CDAShip is limited to the encryption algorithms provided by the Java Cryptographic Extension (JCE) and the Java Cryptographic Architecture (JCA). However, the most common algorithms are included and can be used. For digitally signing the documents, the Apache package is used and therewith the digital signature algorithms are restricted to the ones supported in the package. Another limitation that results from implementation choices is that the XACML policies are restricted to the features that are provided in Sun's XACML implementation, since this is used to evaluate the policies.

Not yet implemented but still considered in our approach is the extension of the key share life time, which was discussed in Section 3.4.3. However, the expiry time is included in the key share information document and the CDAShip can easily be extended to check whether a key share has expired before requesting the key shares from other caregivers. If the key share has expired the needed information has to be requested from the caregiver that originally sent the document. Also, the automatic deletion of expired key shares has not yet been implemented.

Another limitation of the current CDAShip is that the documents are encrypted using the caregivers public key and then distributed, e.g. via email, to the corresponding caregiver. The design requires that the Transport component implements a deleteMessage method. This method is called by the Import/Export component prior to processing of any received document. It is further used to ensure that the caregiver or even worst an unauthorized person does not have access to the received document outside the CDAShip. This may be a security issue, because it may be possible to access the message and therewith the documents before the Import/Export component can delete the message. A possible solution for this problem could be to assign a public and private key pair to each instance of a CDAShip. This key pair can be automatically created when the CDAShip is initialized, which would ensure that the private key is only known to the CDAShip and cannot be abused. The key pair from the caregiver's CDAShip will be used for encryption and decryption.

Also it may be useful to implement a mechanism that keeps track of which key shares have already been requested and from whom in order to prevent replay attacks. Theoretically, the used encryption and signing mechanisms provide authenticity and integrity, but a caregiver could maliciously replay the request.

Another limitation, which is important to mention is that the CDAShip is only secure if the abstract components are implemented in a secure manner. If the FileSystem, for example, does not save the documents in a secured location or if the Transport component does not exchange the documents over a secured channel the CDAShip itself will not be

trustful and secure. Therefore, it is important that the realization of these components is done by a trusted party.

Chapter 7

Related Work

The previous work related to our approach can be divided into the following two categories

- overriding access control policies,
- context based encryption,

The sections below list the work done in these areas.

7.1 Overriding Access Rights

Traditional access control models are restricted and inflexible, because they only permit or deny the access to certain data. They also assume that at the time when polices are written all possible situations are known and can be considered. However in reality this is not a valid assumption, since unanticipated situations may occur. Therefore, in such emergency situations it should be possible to override those policies.

Dean Povey introduced the paradigm of optimistic security in [36]. He describes that a mechanism for overriding access control should be taken into account where it can be assumed that the risk of failure and the cost of recovery is low compared to the cost of not granting access in a given situation. Further, he takes the healthcare domain as an ideal example, since it may be possible that a patient's life depends on the access to information. In addition, he specifies that a system, which implements optimistic security, should have the ability to rollback the actions and to also securely audit them. In our approach it is not possible that the action that has been taken against the actual access rights can be rolled

back, since we only allow read access and once a caregiver reads certain information, it is not possible to delete them out the caregiver's memory. As last requirement, he specifies that a user of the system has to be uniquely identified, so that if the access right has been overridden out of malicious reasons it is possible to make the user responsible for the action. In our approach this is realized by encryption using public keys, that way only the intended caregiver can decrypt the information and therewith it is ensure who accessed it.

Based on Povey's approach Rissanen et al. [14] define the following three possible results of an access request: *permit*, *deny* and *possible-with-override*. Further, they define a combination of these in order to catch even the unanticipated situation, which have not been considered. These combinations are:

- Define the permitted and possible accesses and deny every other situation.
- Define permit and deny accesses and everything else will be possible-with-override.
- Define the denied and possible accesses and permit every other situation.

In our approach we only use the permit and possible-with-override result, since it is assumed that a caregiver should have access to the information in emergency cases. Therefore, an absolute deny is not suitable.

Further, they state that one of the major requirements of a system that allows overriding of policies is the auditing of the actions as well as the creating of a warning message before unauthorized data is accessed. They also recommend XACML as policy langauge since it includes obligations, which can be used for warning and logging.

Directly related to our approach and based on the fact that overriding access control is highly important in the health domain, is the "Break the glass" approach from Ferreira et al. [18]. They have developed a system for the Hospital S. João in Portugal that provides maximum freedom, but also maximum responsibility. This means that the system provides access at all times and whenever it is needed, but that the violation of the access right is strictly audited and the responsible supervisor will be notified. If the action was taken maliciously the user will be taken to responsibility. In order to prevent the user claiming

that the access was done without knowledge of the privacy violation, the system will show a warning message, which has to be explicitly confirmed, and the system will require that the user gives a reason for the access to the information. If either the confirmation or the reason is missing the system will not provide the requested information. The developed system is already used in a hospital with more than 500 doctors, who use it on a daily basis. The difference between this system and our approach is that it is specifically designed for a hospital in which the hierarchy of responsibility can be defined, and in case a user access information against the given right the user's supervisor will be notified. In a peer-to-peer network as described in this thesis, it is not possible to define a supervisor for a doctor. However, in our approach we notify the patient, who then can take legal steps in case the action was malicious.

In general, our approach differs from other approaches in the way that we consider the patient's consent in our policies. Most other systems, such as the one in the Hospital S. Joao, define policies only based on the role of a user and on other organizational reasoning, but seldom on the actual concern of the patient.

7.2 Context-Based Encryption

Hwang and Chang [21] present an element-wise encryption and document signing mechanism, which is able to perform transformation, encryption and signing simultaneously. Furthermore, their approach includes an automatic key downloading mechanism. However, they do not combine the feature of symmetric and public key encryption.

Another approach developed by Bertino and Ferrari [15], who define a formal model of access control policies applied to XML document. The main focus on their approach is encrypting a XML document once with a minimal amount of secret keys and sending it to several recipients, who have different access rights. They apply the policies to the XML document and mark each element with the ID of the policy. Furthermore, they create a secret key for each distinct group of policies, which are applied to the elements and encrypt

them. The key information as well as the encrypted document are stored. Whenever the document has to be shared, the secret keys will be encrypted according to the policies and the already encrypted document will be sent to the recipient. This approach has the advantage of sending the same documents and keys to all the recipients to which the document should be released. However, the CDAShip component does not plan to store documents or keys. Furthermore, the documents are most likely sent to just one recipient at a time.

Chapter 8

Conclusion

8.1 Summary

Electronic information exchange is seen as a means of lowering the cost while increasing the quality of health care. This is potentially realizable as many health care institutions already employ information systems to manage certain aspects of patient care. However, these systems are often disjointed/incompatible. This results in document printing and subsequent exchange with other caregivers via mail, where the other caregivers do not use the same system.

Also information exchange among caregivers, such as doctors, hospitals and laboratories is of highly distributed nature. A caregiver might need to exchange information with another caregiver, who works next door or with caregivers that are located across the world. Therefore, a system that realizes medical information exchange has to be flexible, scalable and decentralized, such as a peer-to-peer network.

In addition to the requirement for flexibility, scalability and decentralization, it is most important to secure the exchanged data, since medical data is highly sensitive. As an aspect of security, the patient's consent should be ensured by taken it into account prior to exchange. This can be done by evaluation policies, which contain patient, organizational and legislative concerns.

This thesis presented a security and privacy mechanism, called CDAShip, for exchanging medical data in the form of CDA documents over a peer-to-peer network. The mech-

anism uses XACML policy evaluation to determine which parts of the information is not allowed to be exchanged. Further, unauthorized information is encrypted to ensure that the receiver is not able to read this information. However, there can be potential conflicts between the patient's right for privacy and the right to receive treatments from well-informed caregivers. In case of such conflict between patient's privacy and patient's safety, the patient's safety should have precedence. Therefore, in emergency cases a caregiver should have the ability to override the patient's privacy policies on behalf of the patient and access the needed information. For this reason the CDAShip exchanges the key used to encrypt the unauthorized sections by splitting it into n pieces, which are then distributed among caregivers. If the unauthorized information needs to be accessed, it is possible to request the key shares and reconstruct the key. It is important to mention that only t out of the n shares are needed to reconstruct the key in order to ensure that the information can be accessed even if some caregivers are not able to provide their key share. However, in order to avoid the misuse of the override feature, the auditing and reporting of the access of unauthorized information is important. Therefore, for such cases the CDAShip will inform the patient as well as log the action.

In order to allow the integration of the CDAShip with different systems, it has abstract components for auditing, transport as well as saving the documents to a file system. These abstract components need to be implemented for the integration with different systems, which makes the CDAShip highly flexible.

The CDAShip securely exchanges XML documents and all the documents that it produces are also encoded in XML, therefore it cannot only be used within the health domain but it can also be applied to any domain where documents are based on XML. In this thesis we applied the CDAShip to the healthcare domain as an example.

8.2 Contributions

The main contributions of this thesis are:

Automatic policy integration: The developed CDAShip automatically interprets privacy policies prior to the information exchange and secures the documents according to the policy evaluation result. As to the best of our knowledge there is currently no implementation of a medical peer-to-peer medication system, which automatically interprets policies prior to the information exchange.

Overriding of policies in emergency cases: As motivated in this thesis it is necessary to allow the override of the privacy policies in emergency cases, since it is safer for the patient to be treated from a caregiver that has the best knowledge.

Non-monotonic information exchange: By exchanging the policies along with the secured documents the CDAShip allows to exchange the information in a non-monotonic way. This means that caregivers with less access rights are able to exchange information that they have received in encrypted form to caregivers with more access rights in decrypted form.

Open Source prototype implementation: The current implementation of the CDAShip is published as a open source project and available online at `http://www.ppci.ca/CDAShip`.

8.3 Future Work

The CDAShip is still in its first stage of development and a first prototype has been implemented. However, parts of the current implementation are only used as a proof of concept, therefore a security analysis to enhance the security of the current proof of concept implementation and to identify other vulnerabilities in the approach would have to be the next step for this project. After the security hardening it would be necessary integrate the CDAShip with an existing EMR system and test its usability.

Also, combining the flexibility of the peer-to-peer network with availability of a central repository should be explored. This would mean that the information can not only be exchanged with various caregivers all over the world, but within a local region it could also be possible to store the information on a central server, in order to ensure availability.

Bibliography

[1] Health Level 7 (HL7). http://www.hl7.org/about/. Acessed October 4, 2006.

[2] Primary Health Care Renewal Electronic Medical Summary Project (e-MS) Core Data Set. http://www.e-ms.ca/documents/Core%20Data%20Set.pdf, December 17, 2004. Acessed March 11, 2005.

[3] Carlisle Adams and Steve Lloyd. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley, 2nd edition, November 6 2002.

[4] Paul Ashley, Satoshi Hada, Günter Karjoth, and Matthias Schunter. E-P3P Privacy Policies and Privacy Authorization. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*, pages 103–109. ACM Press, 2002.

[5] Jr Barrows, RC and PD Clayton. Privacy, confidentiality, and electronic medical records. *J Am Med Inform Assoc*, 3(2):139–148, 1996.

[6] Josh Cohen Benaloh. Secret sharing homomorphisms: keeping shares of a secret secret. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 251–260, London, UK, 1987. Springer-Verlag.

[7] I. Bilykh, Y. Bychkov, D. Dahlem, J. H. Jahnke, G. McCallum, C. Obry, A. Onabajo, and C. Kuziemsky. Can grid services provide answers to the challenges of national health information sharing? In *CASCON '03: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*, pages 39–53. IBM Press, 2003.

[8] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley, November 2002.

[9] G. R. Blakley. Safeguarding cryptographic keys. pages 313–317, 1979.

[10] John Boyer. Canonical XML. http://www.w3.org/TR/xml-c14n, 15 March 2001. W3C Recommendation, Accessed Octorber 3, 2006.

[11] Mark L. Rangappa S. Coppola M.N., Burke D.E. HIPAA Awareness and Preparedness: national survey of Longlifephysician practices. *Group Practice Journal*, 51(5):82 – 83, May 2002.

[12] Robert H. Dolin, Liora Alschuler, Calvin Beebe, Paul V. Biron, Sandra Lee Boyer, Daniel Essin, Elliot Kimber, Tom Lincoln, and John E. Mattison. The HL7 Clinical Document Architecture. *J Am Med Inform Assoc*, 8(6):552–569, 2001.

[13] Robert H. Dolin, Liora Alschuler, Sandy Boyer, Calvin Beebe, Fred M. Behlen, Paul V. Biron, and Amnon Shabo (Shvo). HL7 Clinical Document Architecture, Release 2. *J Am Med Inform Assoc*, 13(1):30–39, 2006.

[14] B. S. Firozabadi E. Rissanen and M. Sergo. Towards a mechanism for discretionary overriding of access control. *12th International Workshop on Security Protocols, Cambridge, UK*, 2004.

[15] Elena Ferrari Elisa Bertino. Secure and selective dissemination of XML documents. *ACM Transactions on Information and System Security*, 5(3):290–331, August 2002.

[16] EPAL. Enterprise Privacy Authorization Language (EPAL 1.2) W3C Member Submission 10 November 2003. `http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/`. Accessed October 3, 2006.

[17] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. pages 427–437, 1987.

[18] A. Ferreira, R. Cruz-Correia, L. Antunes, P. Farinha, E. Oliveira-Palhares, D.W. Chadwick, and A. Costa-Pereira. How to break access control in a controlled manner. In *Proceedings of the 19th IEEE International Symposium on Computer-Based Medical Systems*, pages 847–851, June 2006.

[19] Policy Research Group. The ponder policy based management toolkit. `http://www-dse.doc.ic.ac.uk/Research/policies/ponder/PonderSummary.pdf`. Accessed Octorber 4, 2006.

[20] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or: How to cope with perpetual leakage. pages 339–352, 1995.

[21] Gwan-Hwan Hwang and Tao-Ku Chang. An operational model and language support for securing XML documents. *Computers & Security*, 23(6):498–529, September 2004.

[22] Canada Health Infoway. EHRS Blueprint an interoperable EHR framework. `http://knowledge.infoway-inforoute.ca/`. Accessed Octorber 4, 2006.

[23] R. Deng J. Zhou. On the validity of digital signatures. *ACM SIGCOMM Computer Communication Review*, 30(2):29 – 34, 2000.

[24] E. Jacksch. PIPEDA: Privacys Final Frontier? *Monitor Magazine Online*, 11(4):82 – 83, 2003.

[25] Armin Liebl. Authentication in Distributed Systems: A Bibliography. *SIGOPS Oper. Syst. Rev.*, 27(4):31–41, 1993.

[26] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah. First Experiences using XACML for Access Control in Distributed Systems. In *Proceedings of the 2003 ACM workshop on XML security*, pages 25–37. ACM Press, 2003.

[27] E. C. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[28] B. Fox B. LaMacchia M. Bartel, J. Boyer and E. Simon. XML-Signature Syntax and Processing . `http://www.w3.org/TR/xmldsig-core/`, 12 February 2002. W3C Recommendation, Accessed Octorber 4, 2006.

[29] A. J. (Alfred J.) Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*. The CRC Press series on discrete mathematics and its applications. 1997.

[30] SUN Microsystems. J2ee javamail. `http://java.sun.com/products/javamail/`. Accessed August 5, 2006.

[31] SUN Microsystems. Java cryptography extension (jce) for the java 2 sdk, v 1.4. `http://java.sun.com/products/jce/index-14.html`. Accessed August 5, 2006.

[32] SUN Microsystems. Sun's XACML Implementation. `http://sunxacml.sourceforge.net/index.html`. Accessed August 20, 2006.

[33] SUN Microsystems. Java Cryptography Architecture API Specification and Reference. `http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html`, August 2002. Accessed August 22, 2006.

[34] Jahnke J. H. Onabajo A., Bilykh I. Wrapping Legacy Medical Systems for Integrated Health Network. *In Proceedings 2003 NET.ObjectDays Conference*, pages 544 – 551, 2003.

[35] P3P. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification W3C Working Draft 20 July 2004. `http://www.w3.org/TR/2006/WD-P3P11-20060210/`. Accessed October 4, 2006.

[36] Povey. Optimistic security: A new access control paradigm. In *WNSP: New Security Paradigms Workshop*. ACM Press, 2000.

[37] Bruce Schneier. *Applied cryptography : protocols, algorithms, and source code in C*. Wiley, New York, 2nd edition, 1996. Bruce Schneier.; Includes bibliographical references (p. 675-741) and index.

[38] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[39] Ed Simon. XML Encryption: Applications & Proposal Overview. `http://www.w3.org/2000/11/02-xml-encryption-ws/simon/`. Accessed Octorber 4, 2006.

[40] B. Dillaway T. Imamura and E. Simon. XML Encryption Syntax and Processing. `http://www.w3.org/TR/xmlenc-core/`, 10 December 2002. W3C Recommendation, Accessed Octorber 4, 2006.

[41] W3C. Extensible Markup Language (XML). `http://www.w3.org/XML/`, 2004. Accessed October 4, 2006.

[42] W3C. Document Object Model (DOM). `http://www.w3.org/DOM/`, 2005. Accessed October 4, 2006.

[43] XACML. eXtensible Access Control Markup Language (XACML) Version 2.0, OASIS Standard 1 February, 2005. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`, 2005. Accessed Jun 7, 2006.

Appendix A

Key Share XML Schema

```
1  <?xml version="1.0" encoding="UTF-8"?>

3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:ks="http://www.ppci.ca/CDAShip/KeyShare#"
       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:cd
       :04" targetNamespace="http://www.ppci.ca/CDAShip/KeyShare#"
       elementFormDefault="qualified">
4      <xs:import namespace="http://www.w3.org/2001/04/xmlenc#"
          schemaLocation="http://www.cs.uvic.ca/~obry/CDAShip/Schema/
          Redefined.xsd"/>
5      <xs:import namespace="urn:oasis:names:tc:xacml:2.0:context:schema:cd
          :04" schemaLocation="http://docs.oasis-open.org/xacml/
          access_control-xacml-2.0-context-schema-cd-04.xsd"/>
6      <xs:element name="KeyShare" type="ks:KeyShareType"/>

8      <xs:complexType name="KeyShareType">
9        <xs:sequence>
10         <xs:element ref="xenc:EncryptedKey"/>
11         <xs:element name="KeyName" type="xs:string"/>
12         <xs:element name="SecretKeyEncryptionMethod" type="xs:string"/>
13         <xs:element ref="ks:KeyShareInfo"/>
14       </xs:sequence>
15       <xs:attribute name="Id" type="xs:string" use="required"/>
16     </xs:complexType>

18     <xs:element name="KeyShareInfo">
19       <xs:complexType>
20         <xs:sequence>
21           <xs:element name="ShareNumber" type="xs:int"/>
22           <xs:element name="Threshold" type="xs:int"/>
23           <xs:element name="TotalShares" type="xs:int"/>
24           <xs:element name="Prime" type="xs:decimal"/>
25           <xs:element name="NegativeSecret" type="xs:boolean"/>
```

```
26          <xs:element ref="ks:ExpiryDate"/>
27          <xs:element ref="ks:ShareHolders"/>
28        </xs:sequence>
29      </xs:complexType>
30    </xs:element>

32    <xs:element name="ExpiryDate">
33      <xs:complexType>
34        <xs:sequence>
35          <xs:element name="Month" type="xs:int"/>
36          <xs:element name="Day" type="xs:int"/>
37          <xs:element name="Year" type="xs:int"/>
38        </xs:sequence>
39      </xs:complexType>
40    </xs:element>

42    <xs:element name="ShareHolders">
43      <xs:complexType>
44        <xs:sequence>
45          <xs:element ref="ks:ShareHolder" maxOccurs="unbounded"/>
46        </xs:sequence>
47      </xs:complexType>
48    </xs:element>

50    <xs:element name="ShareHolder">
51      <xs:complexType>
52        <xs:sequence>
53          <xs:element name="Address" type="xs:string"/>
54          <xs:element name="ID" type="xs:string"/>
55          <xs:element ref="ks:PublicKeyInfo"/>
56        </xs:sequence>
57      </xs:complexType>
58    </xs:element>

60    <xs:element name="PublicKeyInfo">
61      <xs:complexType>
62        <xs:sequence>
63          <xs:element name="PublicKeyLocation" type="xs:anyURI"/>
64          <xs:element name="PublicKeyType" type="xs:string"/>
65        </xs:sequence>
66      </xs:complexType>
67    </xs:element>

69    <xs:element name="SigKeyInfo">
70      <xs:complexType>
71        <xs:sequence>
72          <xs:element ref="xenc:EncryptedKey"/>
73          <xs:element name="KeyName" type="xs:string"/>
74          <xs:element name="SecretKeyEncryptionMethod"
              type="xs:string"/>
```

```
75          </xs:sequence>
76          <xs:attribute name="Id" type="xs:string" use="required"/>
77        </xs:complexType>
78      </xs:element>
79  </xs:schema>
```

Appendix B

Key Share Request XML Schema

```
1  <?xml version="1.0" encoding="UTF-8"?>

3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:ksr="http://www.ppci.ca/CDAShip/KeyShareRequest#"
       xmlns:ks="http://www.ppci.ca/CDAShip/KeyShare#"
       xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:cd
       :04" targetNamespace="http://www.ppci.ca/CDAShip/KeyShareRequest#"
       elementFormDefault="qualified">
4       <xs:import namespace="http://www.ppci.ca/CDAShip/KeyShare#"
           schemaLocation="http://www.cs.uvic.ca/%7Eobry/CDAShip/Schema/
           KeyShare.xsd"/>
5       <xs:import namespace="urn:oasis:names:tc:xacml:2.0:context:schema:cd
           :04" schemaLocation="http://docs.oasis-open.org/xacml/
           access_control-xacml-2.0-context-schema-cd-04.xsd"/>

7       <xs:element name="KeyShareRequest">
8           <xs:complexType>
9               <xs:sequence>
10                  <xs:element ref="ksr:Requester"/>
11                  <xs:element ref="ksr:RequestFor"/>
12                  <xs:element ref="ksr:Reason"/>
13                  <xs:element ref="xacml-context:Response"/>
14              </xs:sequence>
15          </xs:complexType>
16      </xs:element>

18      <xs:element name="Requester">
19          <xs:complexType>
20              <xs:sequence>
21                  <xs:element ref="ks:ShareHolder"/>
22              </xs:sequence>
23          </xs:complexType>
24      </xs:element>

26      <xs:element name="RequestFor">
27          <xs:complexType>
28              <xs:sequence>
```

```
29                  <xs:element name="CDA_ID" type="xs:string"/>
30                  <xs:element name="KeyShareID" type="xs:string"/>
31              </xs:sequence>
32          </xs:complexType>
33      </xs:element>

35      <xs:element name="Reason">
36          <xs:complexType>
37              <xs:sequence>
38                  <xs:element name="ReasonReason" type="xs:string"/>
39                  <xs:element ref="ksr:ForwardTo" minOccurs="0"/>
40              </xs:sequence>
41          </xs:complexType>
42      </xs:element>

44      <xs:element name="ForwardTo">
45          <xs:complexType>
46              <xs:sequence>
47                  <xs:element ref="ks:ShareHolder"/>
48              </xs:sequence>
49          </xs:complexType>
50      </xs:element>
51  </xs:schema>
```

Appendix C

Redefined XML Encryption Schema

```
1  <?xml version="1.0" encoding="UTF-8"?>

3  <schema xmlns="http://www.w3.org/2001/XMLSchema"
       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       targetNamespace="http://www.w3.org/2001/04/xmlenc#"
       elementFormDefault="qualified" version="1.0">
4      <import namespace="http://www.w3.org/2000/09/xmldsig#"
          schemaLocation="http://www.cs.uvic.ca/~obry/CDAShip/Schema/
          RedefinedSig.xsd"/>
5      <redefine schemaLocation="http://www.w3.org/TR/xmlenc-core/xenc-
          schema.xsd">

7        <complexType name="CipherDataType">
8          <complexContent>
9            <restriction base="xenc:CipherDataType">
10             <choice>
11               <element name="CipherValue" type="base64Binary"/>
12               <element ref="xenc:CipherReference" minOccurs="0"
                    maxOccurs="0"/>
13             </choice>
14           </restriction>
15         </complexContent>
16       </complexType>

18       <complexType name="EncryptionMethodType">
19         <complexContent>
20           <restriction base="xenc:EncryptionMethodType">
21             <sequence>
22               <element name="KeySize" type="xenc:KeySizeType"
                    minOccurs="0" maxOccurs="0"/>
23               <element name="OAEPparams" type="base64Binary"
                    minOccurs="0" maxOccurs="0"/>
24               <any namespace="##other" minOccurs="0" maxOccurs="0"/>
25             </sequence>
26           </restriction>
27         </complexContent>
```

```
28      </complexType>

30      <complexType name="EncryptedKeyType">
31        <complexContent>
32          <restriction base="xenc:EncryptedKeyType">
33            <sequence>
34              <sequence>
35                <element name="EncryptionMethod"
                     type="xenc:EncryptionMethodType" minOccurs="0"/>
36                <element ref="ds:KeyInfo"/>
37                <element ref="xenc:CipherData"/>
38                <element ref="xenc:EncryptionProperties" minOccurs="0"
                     maxOccurs="0"/>
39              </sequence>

41              <sequence>
42                <element ref="xenc:ReferenceList" minOccurs="0"
                     maxOccurs="0"/>
43                <element name="CarriedKeyName" type="string"/>
44              </sequence>
45            </sequence>
46          </restriction>
47        </complexContent>
48      </complexType>

50    </redefine>
51  </schema>
```

Appendix D

Redefined XML Signature Schema

```
1  <?xml version="1.0" encoding="UTF-8"?>

3  <schema xmlns="http://www.w3.org/2001/XMLSchema"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       targetNamespace="http://www.w3.org/2000/09/xmldsig#"
       elementFormDefault="qualified" version="1.0">
4      <redefine schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core
          -20020212/xmldsig-core-schema.xsd">
5        <complexType name="KeyInfoType">
6          <complexContent>
7            <restriction base="ds:KeyInfoType">
8              <choice maxOccurs="unbounded">
9                <element ref="ds:KeyName"/>
10               <element ref="ds:KeyValue" minOccurs="0" maxOccurs="0"/>
11               <element ref="ds:RetrievalMethod" minOccurs="0"
                     maxOccurs="0"/>
12               <element ref="ds:X509Data" minOccurs="0" maxOccurs="0"/>
13               <element ref="ds:PGPData" minOccurs="0" maxOccurs="0"/>
14               <element ref="ds:SPKIData" minOccurs="0" maxOccurs="0"/>
15               <element ref="ds:MgmtData" minOccurs="0" maxOccurs="0"/>
16               <any namespace="##other" processContents="lax"
                     minOccurs="0" maxOccurs="0"/>
17             </choice>
18             <attribute name="Id" type="ID" use="optional"/>
19           </restriction>
20         </complexContent>
21       </complexType>
22     </redefine>
23 </schema>
```

Appendix E

CDAShip Component Class Descriptions and Diagrams

This Appendix lists the classes of each CDAShip component and shows their relationships in class diagrams. The component interfaces are neglected at this point, since they have been introduced in Section 5.2.1.

E.1 Import/Export

E.1.1 Class Description

The Import/Export component consists of the following classes

Export implements the IExport interface and handles the export of documents (see Section 5.2.1.1).

ForwardProperties stores the information about a document that should be forwarded such as information about the person that should receive the document or the shareholders who are supposed to receive the key shares from the secret key used to encrypt the forwarded document.

GenerateKeyShareRequest used to generate the key share request document.

Import implements the IImport interface and handles the processing of the received documents (see Section 5.2.1.1).

ImportExportUtil contains a method that is often used by both, Export and Import class, to created the signature key information document.

KeyShareInfo contains the information needed to collect and process the requested key shares and decrypt a CDA document or it's sections. This information includes the key share ID, the CDA document, a list of sections that should be decrypted, the threshold value to indicate how many key shares are needed to reconstruct the secret key and a list of the already received key shares, which are needed to re-encrypt the document.

LoadPolicy used to load the policies in order to exchange them together with the CDA documents. It is important to mention that if a policy set refers to a different policy that referred policy will also be exchanged and that the location of the policy will be substituted with the ID of the policy. When importing and saving the policies the ID has to be substituted with the policy location again.

DocumentInfo implements the IDocumentInfo interface (see Section 5.2.1.1).

RequestInfo contains information about a request.

InsistRequestEvent is the event, which is thrown in case the caregiver insists on accessing a unauthorized section.

InsistRequestListener listens to the InsistRequestEvents.

E.1.2 Class Diagram

Figure E.2 and E.1 show the class diagram of the Import/Export component.

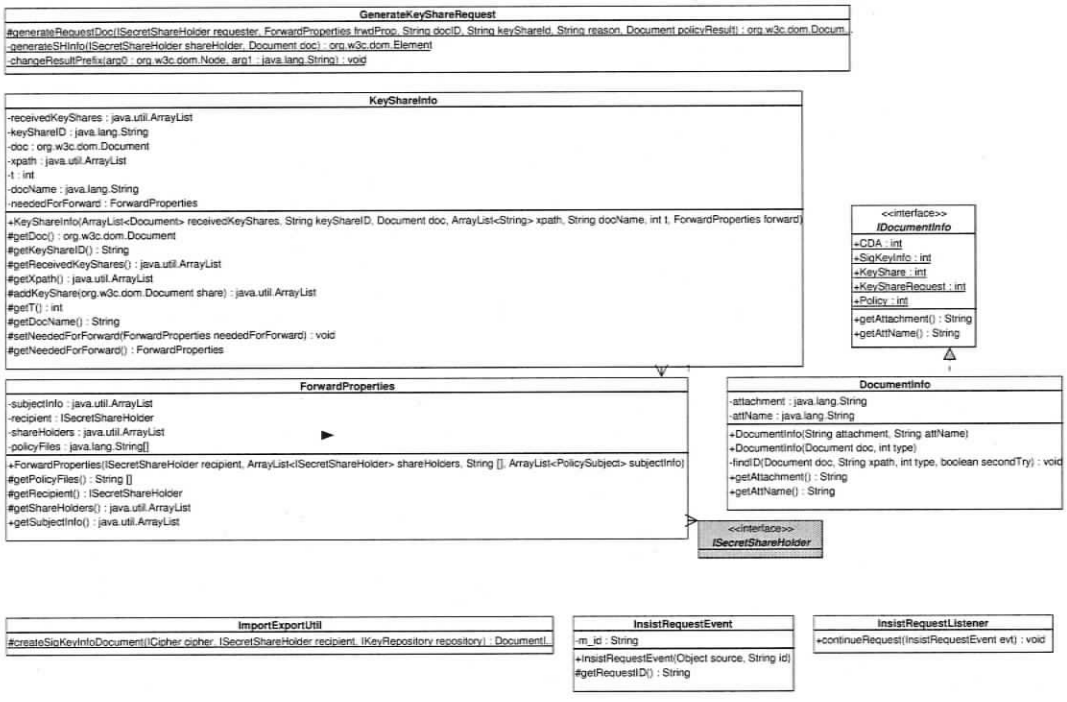

Figure E.1. *Import/Export class diagram, Part 1*

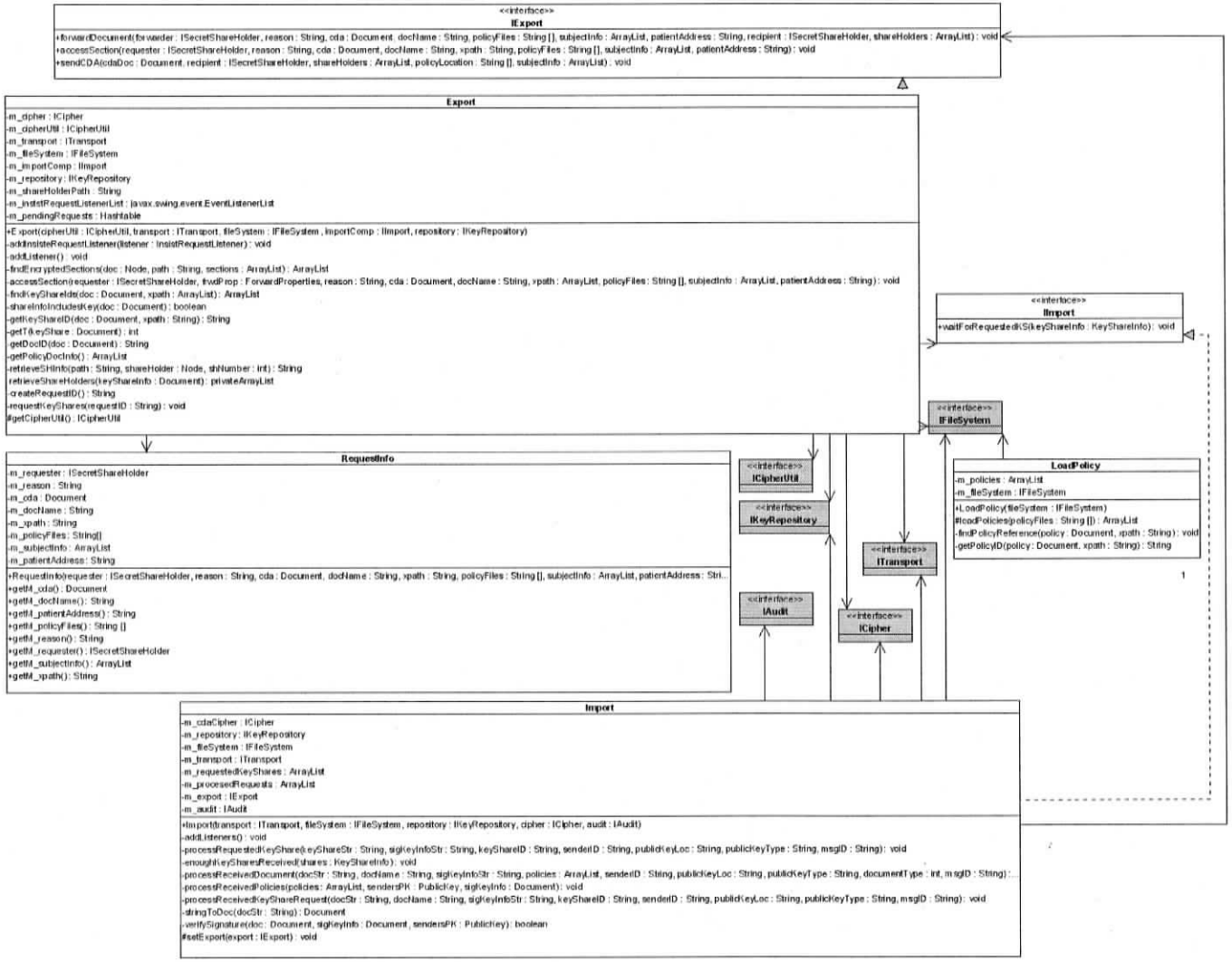

Figure E.2. *Import/Export class diagram, Part 2*

E.2 CDACipher

E.2.1 Class Description

The CDACipher component consists of the following classes

CDACipher handles all the interaction to the component. It is the only class with implements the ICipher interface (see Section 5.2.1.2).

Encryption deals with the encryption of the sections and their attributes.

Decryption handles the decryption of the sections and their attributes.

CreateSignature creates the XML Signature and attaches it to the document.

VerifySignature verifies the signature included in a given XML document.

GenerateSecretKeyInfo generates the document, which contains the secret key information. It also encrypt the secret keys using public key cryptography.

RetrieveSecretKeys retrieves the secret key information from a given document and decrypts the keys using public key cryptography. It also decrypts the digital signature.

SecretShareHolder contains information to identify a shareholder. The information include shareholder ID and address, public key location and type, as well as the public key object.

CipherUtil incudes the settings needed to cipher data. This includes for example the encryption algorithm, the signature method, a list of the shareholders that will receive a key share and more. This class implements the ICipherUtil interface (see Section 5.2.1.2).

E.2.2 Class Diagram

Figure E.3 shows the class diagram of the CDACipher component.

E.3 Policy

E.3.1 Class Description

The Policy component consists of the following classes.

RequestResponse implements the IPolicy interface (see Section 5.2.1.3). This class is responsible for creating the request document and triggering the policy evaluation by using the Sun XACML implementation.

SubjectContent used to create the attributes for the subject in the request document.

ResolveAllCollectObligation defines the policy combining algorithm used in the CDAShip (see Section 5.3.3.1). It defines that all policies have to evaluated and that if a policy results in deny it will collect it's obligations.

CDAShipPDP is the implementation of the Policy Decision Point of the CDAShip. It also adds the resolve-all-and-collect-obligations policy combining algorithm.

Figure E.3. *CDACipher class diagram*

E.3.2 Class Diagram

Figure E.4 shows the class diagram of the Policy component.

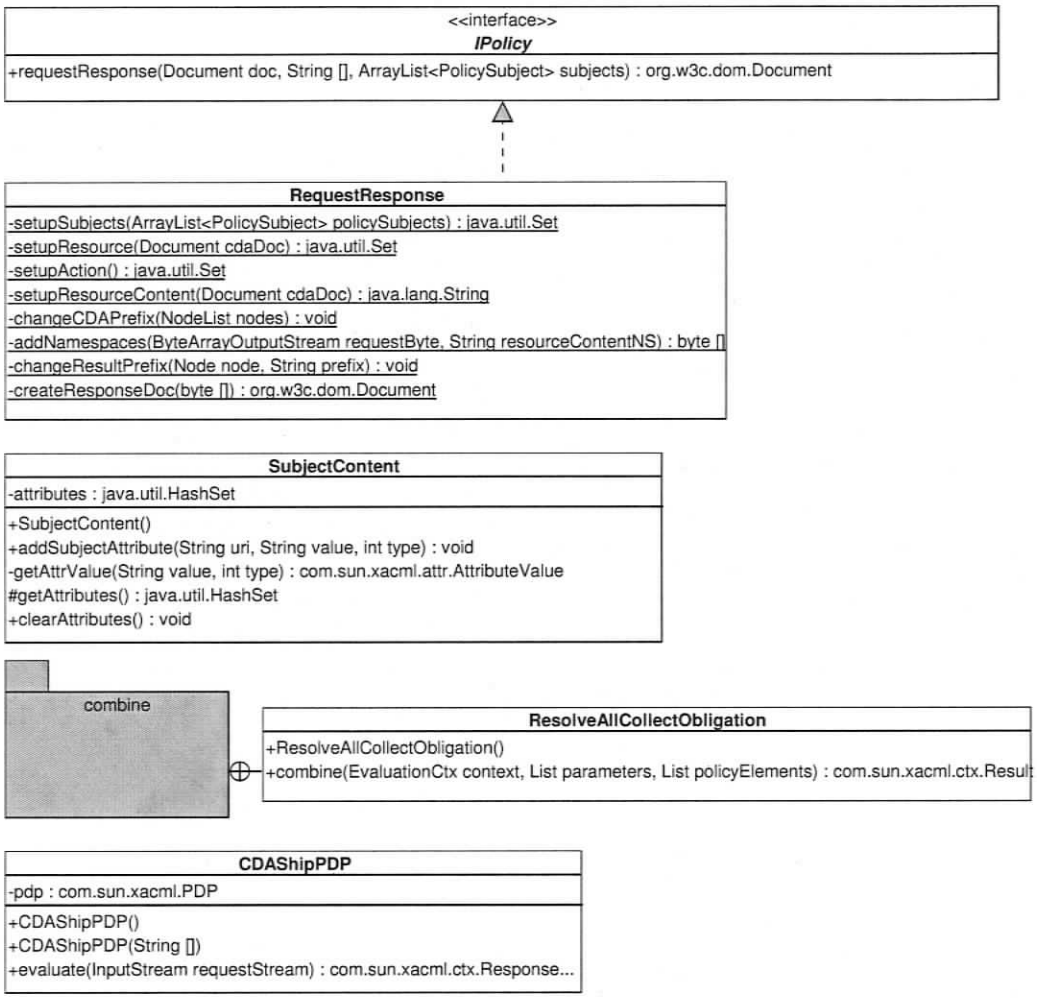

Figure E.4. *Policy class diagram*

E.4 KeyManager

E.4.1 Class Description

The KeyManager component consists of the following classes

Share stores information about a key share such as the share value, the threshold value and the total amount of key shares created. A Share object can be created from a key share

information document or by passing the need parameters. This class implements the IShare interface (see Section 5.2.1.4).

ShareUtil implements the IKeyManager interface and defines how to split and recover a secret.

E.4.2 Class Diagram

Figure E.5 shows the class diagram of the KeyManager component.

E.5 KeyRepository

E.5.1 Class Description

The KeyRepository component consists of the following classes

Repository implements the interface IKeyRepository (see Section 5.2.1.5). The Repository contains methods to load private and public PGP keys and X.509 certificates. It also allows to add other certificate types.

GetEncodedKeySpec is an abstract class. An EncodedKeySpec is needed to create a secret key from a certificate. In case a not yet supported certificate type is added to the repository, it has to be ensured that a getEncodedKeySpec method is implemented.

LoadKeys used to load the keys from a given location.

PrivateKeyProperties contains the properties of a private key.

E.5.2 Class Diagram

Figure E.6 shows the class diagram of the KeyRespository component.

E.6 Transport

E.6.1 Class Description

The Transport component consists of the following classes. It is important to mention that each Event also has an EventListener. The EventListeners are not listed here.

ReceivedEvent is the basic event and stores the sender's information such as the public key location and type or the sender's ID as well as the received document. All other received events extend this event.

DocumentReceivedEvent is thrown when a CDA document or a key share arrives and contains further information about the received document such as the document type and the signature key information document.

KeyShareRequestReceivedEvent is thrown when a key share request arrives. In addition to the information from the ReceivedEvent it also contains the ID of the secret key from which the share is requested.

RequestedKeyShareReceivedEvent is thrown when a requested key share was received. It additionally contains the ID of the secret key whose share was received as well as the signature key information document.

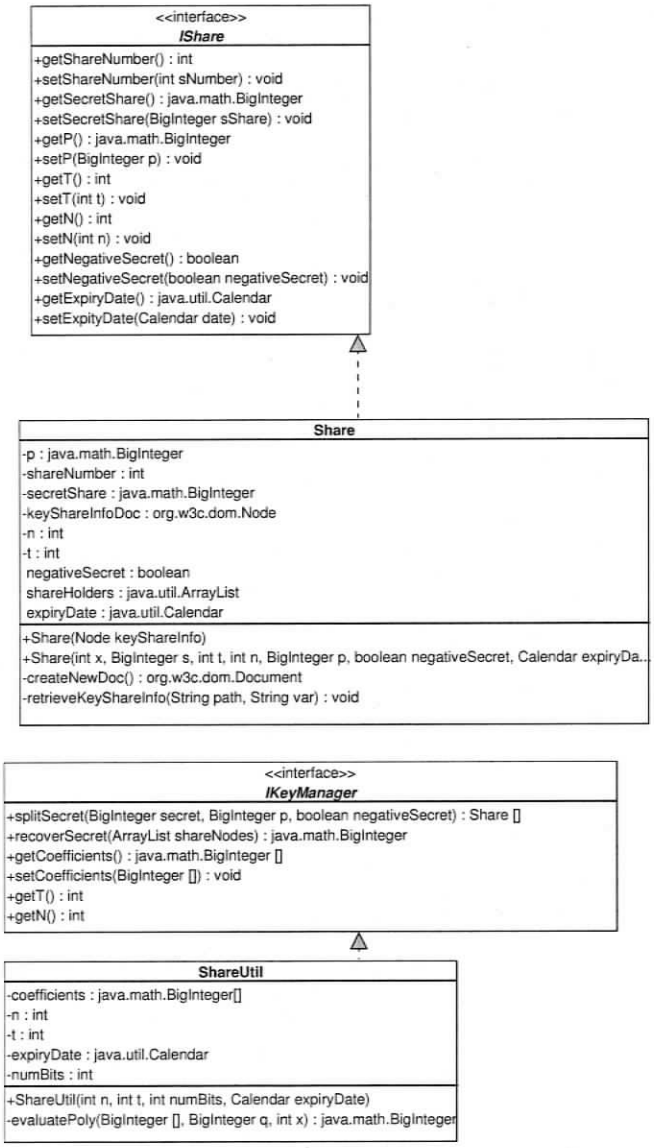

Figure E.5. *KeyManager class diagram*

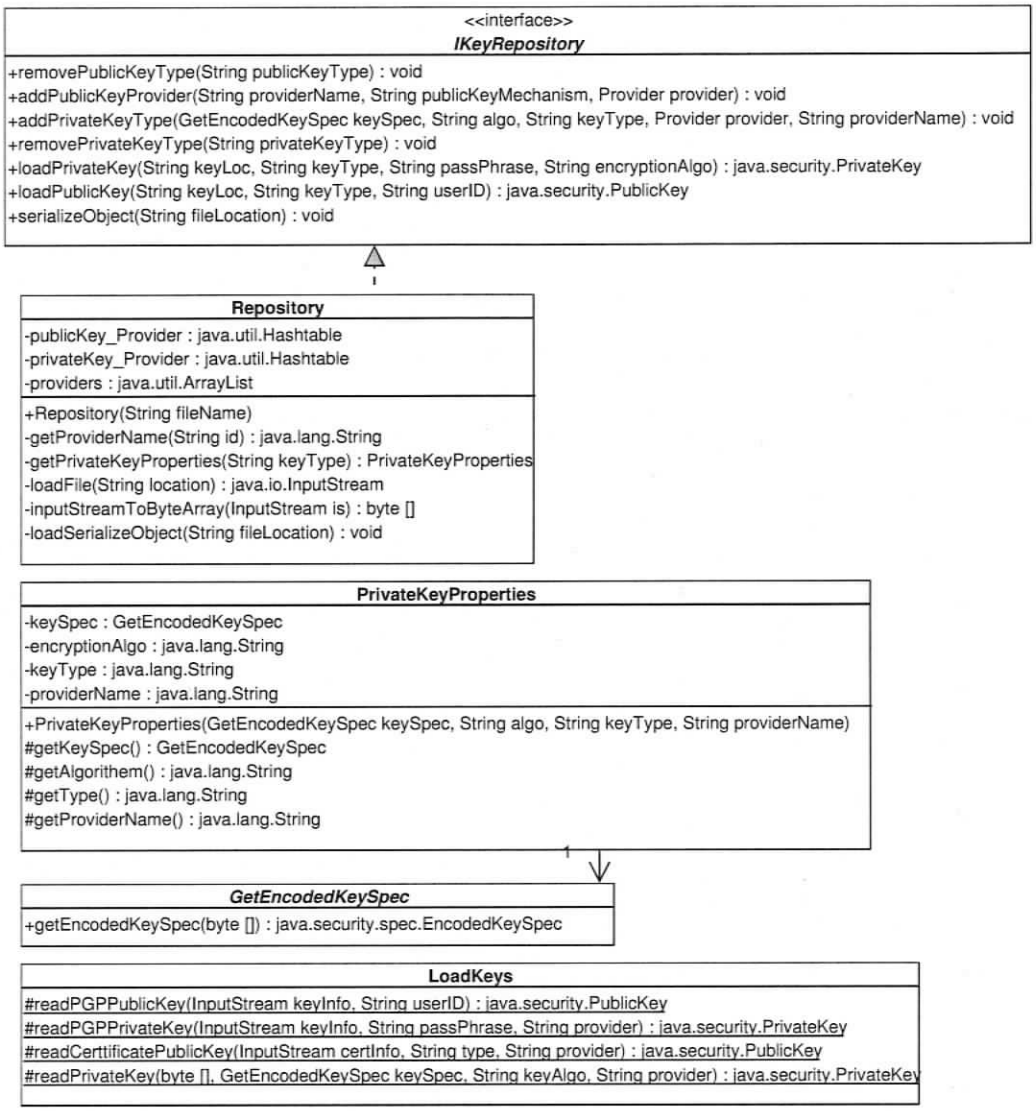

Figure E.6. *KeyRespository class diagram*

E.6.2 Class Diagram

Figure E.7 shows the class diagram of the Transport component.

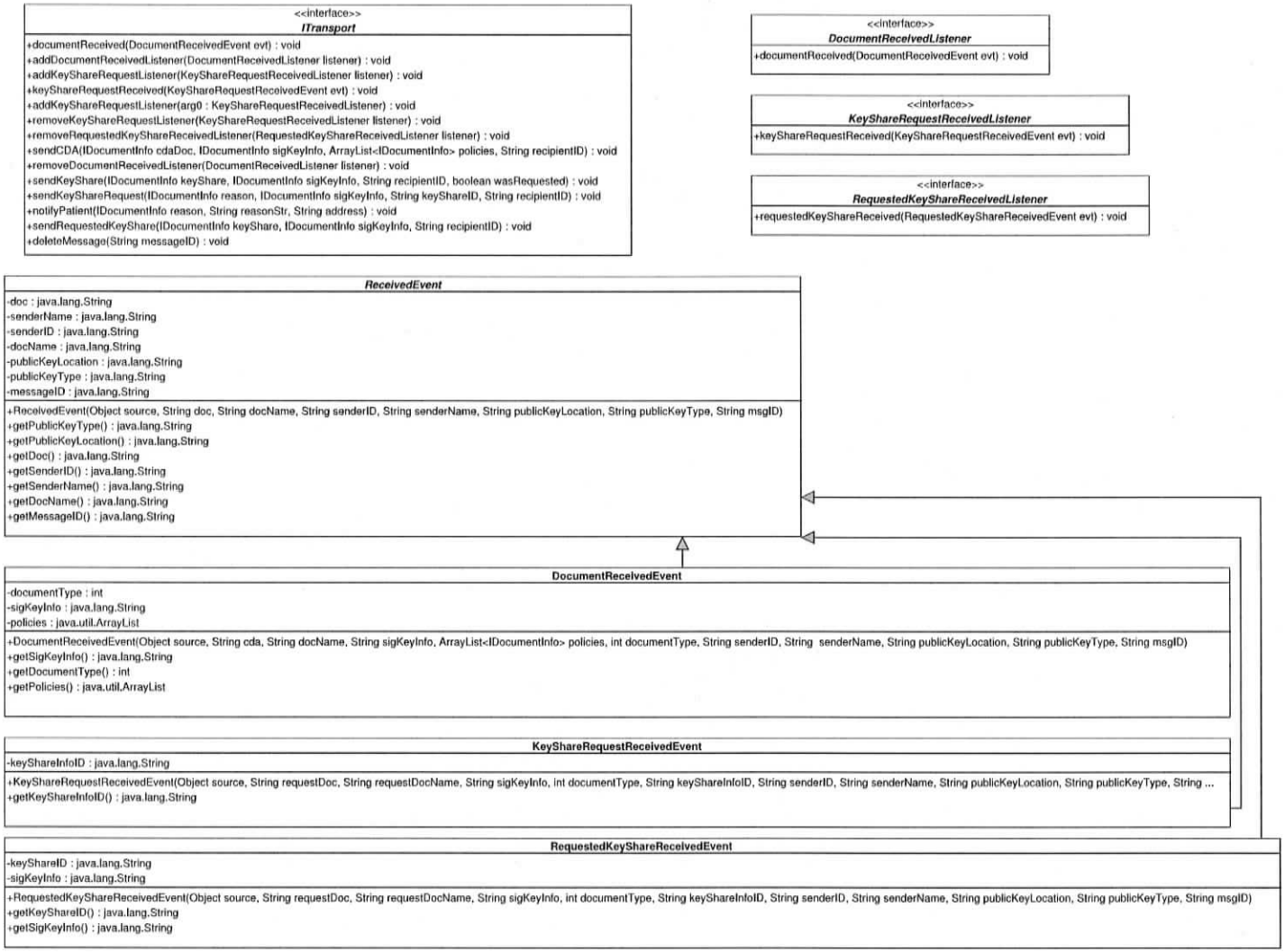

Figure E.7. *Transport class diagram*

E.7 Audit

E.7.1 Class Description

The Audit component only consists of an interface (see Section 5.2.1.7).

E.7.2 Class Diagram

Figure E.8 shows the class diagram of the Audit component.

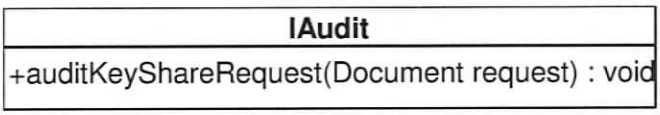

Figure E.8. *Audit class diagram*

E.8 FileSystem

E.8.1 Class Description

The FileSystem component only consists of an interface (see Section 5.2.1.8).

E.8.2 Class Diagram

Figure E.9 shows the class diagram of the FileSystem component.

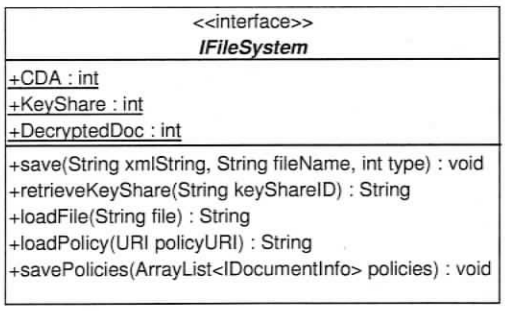

Figure E.9. *FileSystem class diagram*

E.9 Util

E.9.1 Class Description

Besides exception classes, which are self-explanatory, the Util component contains the following two classes.

PolicySubject stores the information about the subject content that is needed in order to create the subject of the XACML request. A subject contains of a type, an URI as well as a value. The information about a subject has to be given by the user of the CDAShip when exchanging documents. It is further passed from the Import/Export component over the CDACipher to the Policy component in order to create the request.

DocumentModification converts DOM document objects to a string object and vice versa.

E.9.2 Class Diagram

Figure E.10 shows the class diagram of the Util component.

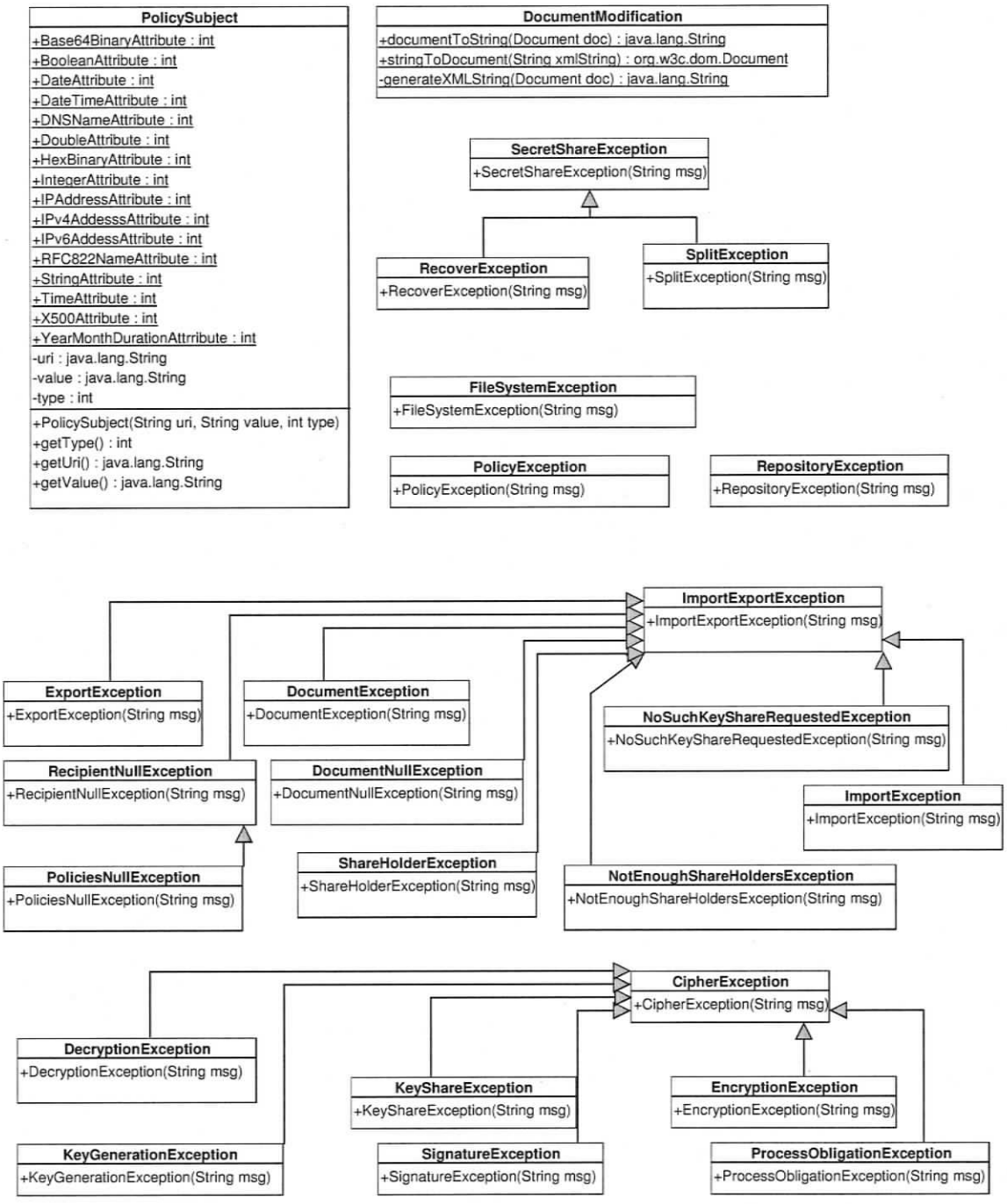

Figure E.10. *Util class diagram*

Appendix F

Sample e-ms CDA document

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ClinicalDocument xmlns:hl7="urn:hl7-org:v3"
       xmlns="urn:hl7-org:v3"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="urn:hl7-org:v3 ../EMS_CDA.xsd">
3  <!-- ***** e-MS CDA Header ************************-->
4   <id extension="DEAF84EC-87F7-4bff-BDB6-9DF00644D80D"
        root="2.16.840.1.113883.3.933"/>
5   <code code="34140-4" codeSystem="2.16.840.1.113883.6.1"
        codeSystemName="LOINC" displayName="Referral"/>
6   <title/>
7   <effectiveTime value="20040809142222"/>
8   <confidentialityCode code="N"
        codeSystem="2.16.840.1.113883.5.25"/>
9   <!-- ***** Information Recipient ************************-->
10  <informationRecipient typeCode="PRCP">
11   <intendedRecipient classCode="ASSIGNED">
12    <id extension="ggottschalk"
          root="DCCD2C68-389B-44c4-AD99-B8FB2DAD1234"/>
13    <informationRecipient>
14     <name>
15      <prefix>Dr.</prefix>
16      <given>Gudrun</given>
17      <family>Gottschalk</family>
18     </name>
19    </informationRecipient>
20    <receivedOrganization>
21     <name>Vancouver General Hospital</name>
22    </receivedOrganization>
23   </intendedRecipient>
24  </informationRecipient>
25  <!-- ***** Author ************************-->
26  <author>
27   <time value="20040812120000"/>
28   <assignedAuthor>
```

```
29     <id extension="hhippocrates"
          root="DCCD2C68-389B-44c4-AD99-B8FB2DAD1493"/>
30     <assignedPerson>
31      <name>
32       <prefix>Dr.</prefix>
33       <given>Harold</given>
34       <family>Hippocrates</family>
35      </name>
36     </assignedPerson>
37    </assignedAuthor>
38   </author>
39   <author>
40    <time value="20040812120000"/>
41    <assignedAuthor>
42     <id extension="hhippocrates"
          root="DCCD2C68-389B-44c4-AD99-B8FB2DAD1493"/>
43     <assignedAuthoringDevice>
44      <softwareName>e-MS Web</softwareName>
45     </assignedAuthoringDevice>
46    </assignedAuthor>
47   </author>
48   <!-- ***** Custodian ************************-->
49   <custodian typeCode="CST">
50    <assignedCustodian>
51     <representedCustodianOrganization>
52      <id extension="123"
           root="7EEF0BCC-F03E-4742-A736-8BAC57180C5F"/>
53      <name>Level 7 Healthcare, Inc</name>
54     </representedCustodianOrganization>
55    </assignedCustodian>
56   </custodian>
57   <!-- ***** Record Target ************************-->
58   <recordTarget typeCode="RCT" contextControlCode="OP">
59    <patient classCode="PAT">
60     <id extension="9999999999"
          root="2BFBA1E9-79C2-4bbb-B589-41B949BD6A3B"
          assigningAuthorityName="BC-PHN"/>
61     <id extension="99999-9" assigningAuthorityName="WCB"/>
62     <addr>
63      <streetAddressLine>2222 Home Street</streetAddressLine>
64      <city>Ann Arbor</city>
65      <state>BC</state>
66      <postalCode>A1B 2C3</postalCode>
67      <country>CA</country>
68     </addr>
69     <patientPatient>
70      <name>
71       <prefix>Mrs.</prefix>
72       <given>Eve</given>
73       <family>Everywoman</family>
```

```
74      </name>
75     </patientPatient>
76    </patient>
77   </recordTarget>
78   <!-- ***** Participant - Contact ************************-->
79   <participant typeCode="NOT">
80    <participatingEntity classCode="CON">
81     <code code="HUSB"
          codeSystem="F05716F7-BA77-4641-A8D8-6E3948CCD321"
          codeSystemName="HL7: PersonalRelationshipRoleType"
          displayName="Husband"/>
82     <addr>
83      <streetAddressLine>2222 Home Street</streetAddressLine>
84      <city>Ann Arbor</city>
85      <state>BC</state>
86      <postalCode>A1B 2C3</postalCode>
87      <country>CA</country>
88     </addr>
89     <telecom value="tel:(555) 555-5001" use="HP"/>
90     <telecom value="mailto: Neville.Nuclear@eg.net" use="HP"/>
91     <associatedPerson>
92      <name>
93       <prefix>Mr.</prefix>
94       <given>Neville</given>
95       <family>Nuclear</family>
96      </name>
97     </associatedPerson>
98    </participatingEntity>
99   </participant>
100  <!-- ***** Related Document ************************-->
101  <relatedDocument typeCode="RPLC">
102   <parentDocument>
103    <id extension="B7330FD6-E812-4b4f-809C-32172EF21812"
          root="2.16.840.1.113883.3.933"/>
104    <code code="34140-4" codeSystem="2.16.840.1.113883.6.1"
          codeSystemName="LOINC" displayName="Referral"/>
105    <text>A previous referral is related to this document</text>
106   </parentDocument>
107  </relatedDocument>
108  <!-- ***** e-MS CDA Structured Body ************************-->
109  <component>
110   <structuredBody>
111    <!-- ***** Purpose ************************-->
112    <component>
113     <section>
114      <code code="001"
           codeSystem="7BA9BFFD-D25F-44e8-A7B0-0DF214D6845B"
           codeSystemName="e-MS Document Section Codes"
           displayName="Purpose"/>
115      <title>Referral Purpose</title>
```

```
116       <text>
117        <paragraph>Please see this patient regarding occasional
              occurrences of arrhythmia at rest accompanied with mild
              shortness of breath. Mild exercise leads to faintness and
               mild chest pain.</paragraph>
118        <paragraph>Urgency: expedite</paragraph>
119        <table border="1">
120         <tbody>
121          <tr>
122           <th>Other Provider</th>
123           <th>Reason</th>
124          </tr>
125          <tr>
126           <td>Dr. Vera V. Valve at Level 7 Healthcare, Inc </td>
127           <td>High blood pressure in the past</td>
128          </tr>
129          <tr>
130           <td>Dr. Sara S. Specialize at Good Health Hospital</td>
131           <td>24 Hour ECG recording</td>
132          </tr>
133         </tbody>
134        </table>
135        <paragraph>Comments: The patient expressed that this problem
              has gradually been getting worse over the course of the
              past year and a half.</paragraph>
136       </text>
137       <entry typeCode="DRIV">
138        <observation classCode="OBS" moodCode="EVN">
139         <code nullFlavor="NA"/>
140         <text>Please see this patient regarding occasional
              occurrences of arrhythmia at rest accompanied with mild
              shortness of breath. Mild exercise leads to faintness
              and mild chest pain. </text>
141         <priorityCode code="E"
              codeSystem="1261620F-C09D-4825-96E4-E808DCFA4D17"
              codeSystemName="eMS_Purpose_Urgency"
              displayName="Expedite (call)"/>
142        </observation>
143       </entry>
144      </section>
145     </component>
146     <!-- ***** Allergies ************************-->
147     <component>
148      <section>
149       <code code="008"
            codeSystem="7BA9BFFD-D25F-44e8-A7B0-0DF214D6845B"
            codeSystemName="e-MS Document Section Codes"
            displayName="Allergies"/>
150       <title>Allergies</title>
151       <text>
```

```
152         <table border="1">
153          <tbody>
154           <tr>
155            <th>Allergy Type</th>
156            <th>Severity</th>
157            <th>Comments</th>
158            <th>Date</th>
159           </tr>
160           <tr>
161            <td>Penicillin</td>
162            <td>Mild</td>
163            <td>Penicillin. Resulting in hives. Patient has not been
                  tested recently to see if this is still an active
                  allergy</td>
164            <td>1965</td>
165           </tr>
166           <tr>
167            <td>Dairy products</td>
168            <td>Moderate</td>
169            <td>Dairy. Resulting in puffy eyes</td>
170            <td>1970-1985</td>
171           </tr>
172          </tbody>
173         </table>
174        </text>
175        <entry typeCode="COMP">
176         <observation classCode="OBS" moodCode="EVN">
177          <code code="DA"
                 codeSystem="2A9DDF65-2465-4cf7-903C-832A33B22F5D"
                 codeSystemName="AllergyType"
                 displayName="Drug Allergy"/>
178          <effectiveTime value="19650516"/>
179          <entryRelationship typeCode="COMP">
180           <observation classCode="OBS" moodCode="EVN">
181            <code code="MI" codeSystem="2.16.840.1.113883.12.128"
                    codeSystemName="AllergySeverity" displayName="Mild"/>
182           </observation>
183          </entryRelationship>
184         </observation>
185        </entry>
186        <entry typeCode="COMP">
187         <observation classCode="OBS" moodCode="EVN">
188          <code code="FA"
                 codeSystem="2A9DDF65-2465-4cf7-903C-832A33B22F5D"
                 codeSystemName="AllergyType"
                 displayName="Food Allergy"/>
189          <effectiveTime value="19700000000000.19850000000000"/>
190          <entryRelationship typeCode="COMP">
191           <observation classCode="OBS" moodCode="EVN">
```

```
192           <code code="MO" codeSystem="2.16.840.1.113883.12.128"
                 codeSystemName="AllergySeverity"
                 displayName="Moderate"/>
193          </observation>
194         </entryRelationship>
195        </observation>
196       </entry>
197      </section>
198     </component>
199     <!-- ***** Family History ************************-->
200     <component>
201      <section>
202       <code code="10157" codeSystem="2.16.840.1.113883.6.1"
            codeSystemName="LOINC" displayName="Family History"/>
203       <title>Family History</title>
204       <text>The father of the patient died of a myocardial
            infarction at the age of 65.</text>
205      </section>
206     </component>
207    </structuredBody>
208   </component>
209  </ClinicalDocument>
```

Appendix G

XACML Policy "Receiving Organization"

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- ********* Policy Receiving Organization ******************-->
3  <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
       xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:
       schema:os"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:
       os http://docs.oasis-open.org/xacml/2.0/access_control-xacml
       -2.0-policy-schema-os.xsd" xmlns:md="urn:hl7-org:v3"
       PolicyId="http://www.cs.uvic.ca/~obry/CDAShip/Policies/
       Rule_ReceivingOrg.xml"
       RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining
       -algorithm:deny-overrides">
4  <!-- ********* PolicyDefault ****************************-->
5   <PolicyDefaults>
6    <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</
        XPathVersion>
7   </PolicyDefaults>
8  <!-- ********* Target *****************************-->
9   <Target/>
10 <!-- ********* Rule Permit ************************************-->
11 <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:RulePermit"
12 Effect="Permit">
13   <Description>Permit the exchange of the clinical document
        without encryption, when the document is send to a caregiver
        that belongs to Level 7 Healthcare, Inc.</Description>
14  </Rule>
15 <!-- ********* Rule Deny ************************************-->
16  <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:SimpleRule1"
       Effect="Deny">
17   <Description>Deny the exchange of the clinical document when it
        is send to an organization other than Level 7 Healthcare, Inc
        . Encrypt the recordTarget element in a way that all
        subelements are encrypted and encrypt all components other
        than the Purpose (e-md code 001) section.</Description>
```

```
18    <Target>
19     <Resources>
20      <Resource>
21  <!-- ********* ResourceMatch namespace *************-->
22        <ResourceMatch
             MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
             equal">
23         <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">urn:
              hl7-org:v3</AttributeValue>
24         <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target
              -namespace"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
25        </ResourceMatch>
26  <!-- ********* ResourceMatch CDA *********************-->
27        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function
             :xpath-node-match">
28         <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
              string">//md:ClinicalDocument</AttributeValue>
29         <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
               DataType="http://www.w3.org/2001/XMLSchema#string"/>
30        </ResourceMatch>
31       </Resource>
32     </Resources>
33  <!-- ********* Actions ************************************-->
34     <Actions>
35      <Action>
36       <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:
             string-equal">
37        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#
              string">read</AttributeValue>
38        <ActionAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-
              id"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
39       </ActionMatch>
40      </Action>
41     </Actions>
42   </Target>
43  <!-- ********* Condition***********************************-->
44   <Condition>
45   <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
46    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string
        -equal">
47     <AttributeValue
         DataType="http://www.w3.org/2001/XMLSchema#string">Victoria
          General Hospital</AttributeValue>
```

```
48      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
           string-one-and-only">
49       <AttributeSelector
            RequestContextPath="//xacml-context:Resource/xacml-context
            :ResourceContent/md:ClinicalDocument/md:
            informationRecipient/md:intendedRecipient/md:
            receivedOrganization/md:name/text()"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
50      </Apply>
51     </Apply>
52    </Apply>
53    </Condition>
54   </Rule>
55  <!-- ********* Obligations ********************************-->
56   <Obligations>
57    <Obligation ObligationId="Encrypt" FulfillOn="Deny">
58     <AttributeAssignment
          AttributeId="message"
          DataType="http://www.w3.org/2001/XMLSchema#string">Encrypt
          patient address in the recordTarget.</AttributeAssignment>
59     <AttributeAssignment
          AttributeId="path"
          DataType="http://www.w3.org/2001/XMLSchema#string">/:
          ClinicalDocument/:recordTarget/:patient/:addr</
          AttributeAssignment>
60    </Obligation>
61    <Obligation ObligationId="Encrypt" FulfillOn="Deny">
62     <AttributeAssignment
          AttributeId="message"
          DataType="http://www.w3.org/2001/XMLSchema#string">Encrypt
          patient phone number.</AttributeAssignment>
63     <AttributeAssignment
          AttributeId="path"
          DataType="http://www.w3.org/2001/XMLSchema#string">/:
          ClinicalDocument/:recordTarget/:patient/:telecom</
          AttributeAssignment>
64    </Obligation>
65    <Obligation ObligationId="Encrypt" FulfillOn="Deny">
66     <AttributeAssignment
          AttributeId="message"
          DataType="http://www.w3.org/2001/XMLSchema#string">Encrypt
          patientPatient.</AttributeAssignment>
67     <AttributeAssignment
          AttributeId="path"
          DataType="http://www.w3.org/2001/XMLSchema#string">/:
          ClinicalDocument/:recordTarget/:patient/:patientpatient</
          AttributeAssignment>
68    </Obligation>
69    <Obligation ObligationId="Encrypt" FulfillOn="Deny">
```

```
70      <AttributeAssignment
           AttributeId="message"
           DataType="http://www.w3.org/2001/XMLSchema#string">Encrypt
           all sections, except of the Purpose section (e-MS code =
           001).</AttributeAssignment>
71      <AttributeAssignment
           AttributeId="path"
           DataType="http://www.w3.org/2001/XMLSchema#string">/:
           ClinicalDocument/:component/:structuredBody/:component/:
           section/:code[@code!=001]/parent::*</AttributeAssignment>
72    </Obligation>
73   </Obligations>
74  </Policy>
```

Appendix H

XACML Policy "Urgent"

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- ********* Policy Urgent ******************-->
3  <Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
       xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:
       schema:os"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:
       os http://docs.oasis-open.org/xacml/2.0/access_control-xacml
       -2.0-policy-schema-os.xsd" xmlns:md="urn:hl7-org:v3"
       PolicyId="http://www.cs.uvic.ca/~obry/CDAShip/Policies/
       Rule_Patient_Name.xml"
       RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining
       -algorithm:deny-overrides">
4  <!-- ********* PolicyDefault ******************-->
5   <PolicyDefaults>
6    <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</
        XPathVersion>
7   </PolicyDefaults>
8  <!-- ********* Target *************************-->
9   <Target/>
10 <!-- ********* Rule Permit ********************-->
11  <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:RulePermit"
       Effect="Permit">
12   <Description>Permit the exchange of clinical document, if it
        does not include patient name.</Description>
13  </Rule>
14 <!-- ********* Rule Deny***********************-->
15  <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:SimpleRule1"
       Effect="Deny">
16   <Description>Deny the exchange of clinical document, if the the
        exchange is not urgent. Encrypt name element.</Description>
17   <Target>
18    <Resources>
19     <Resource>
20 <!-- ********* ResourceMatch namespace ********-->
21      <ResourceMatch
           MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
```

```
          equal">
22         <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">urn:
              hl7-org:v3</AttributeValue>
23         <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:2.0:resource:target
              -namespace"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
24       </ResourceMatch>
25  <!-- ********* ResourceMatch CDA **************-->
26       <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:xpath-node-
            match">
27         <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">//md:
              ClinicalDocument</AttributeValue>
28         <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:resource:xpath"
               DataType="http://www.w3.org/2001/XMLSchema#string"/>
29       </ResourceMatch>
30      </Resource>
31     </Resources>
32  <!-- ********* Actions ************************-->
33     <Actions>
34      <Action>
35       <ActionMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-
            equal">
36         <AttributeValue
              DataType="http://www.w3.org/2001/XMLSchema#string">read</
              AttributeValue>
37         <ActionAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-
              id"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
38       </ActionMatch>
39      </Action>
40     </Actions>
41    </Target>
42  <!-- ********* Condition ********************-->
43    <Condition>
44    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
45    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-
         equal">
46       <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">U</
            AttributeValue>
47       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
            string-one-and-only">
```

```
48        <AttributeSelector
             RequestContextPath="//xacml-context:Resource/xacml-context
             :ResourceContent/md:ClinicalDocument/md:component/md:
             structuredBody/md:component/md:section/md:entry/md:
             observation/md:priorityCode/@code" DataType="http://www.w3
             .org/2001/XMLSchema#string"/>
49       </Apply>
50      </Apply>
51      </Apply>
52     </Condition>
53    </Rule>
54  <!-- ********* Obligations ********************************-->
55    <Obligations>
56     <Obligation ObligationId="Encrypt" FulfillOn="Deny">
57      <AttributeAssignment
           AttributeId="message"
           DataType="http://www.w3.org/2001/XMLSchema#string">Access
           denied, because the the purpose in not urgent. Encrypt name
           .</AttributeAssignment>
58      <AttributeAssignment
           AttributeId="path"
           DataType="http://www.w3.org/2001/XMLSchema#string">/:
           ClinicalDocument/:component/:structuredBody/:component/:
           section/:code[@code!=10157 and @code!=001]/parent::*</
           AttributeAssignment>
59     </Obligation>
60    </Obligations>
61  </Policy>
```

Appendix I

XACML Policy Set

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:
       os http://docs.oasis-open.org/xacml/2.0/access_control-xacml
       -2.0-policy-schema-os.xsd"
       PolicySetId="urn:oasis:names:tc:xacml:2.0:example:policysetid
       :1"
       PolicyCombiningAlgId="policy-combining-alg:resolve-all-deny">
3  <!--********** Description  ************************************-->
4      <Description>Policy Set combining the policy Receiving
          Organization and Urgent.</Description>
5  <!--********** Target *****************************************-->
6      <Target/>
7  <!--********** Policy References ********************************-->
8      <PolicyIdReference>http://www.cs.uvic.ca/~obry/CDAShip/
          Policies/Rule_Receiving_Org.xml</PolicyIdReference>
9      <PolicyIdReference>http://www.cs.uvic.ca/~obry/CDAShip/
          Policies/Rule_Urgent.xml</PolicyIdReference>
10 <!--********** Polciy *****************************************-->
11     <Policy PolicyId="urn:oasis:names:tc:xacml:2.0:example:
          policyid:15" RuleCombiningAlgId="urn:oasis:names:tc:xacml
          :1.0:rule-combining-algorithm:deny-overrides">
12       <Target/>
13     </Policy>
14 </PolicySet>
```