Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 168 (2007) 221–236
www.elsevier.com/locate/entcs

Audit-Based Access Control for Electronic Health Records*

M.A.C. Dekker[a,1] S. Etalle[b,2]

[a] *Security group, TNO ICT, Delft, The Netherlands*

[b] *Distributed and Embedded Systems Group, University of Twente, Enschede, The Netherlands*

Abstract

Traditional access control mechanisms aim to prevent illegal actions a-priori occurrence, i.e. before granting a request for a document. There are scenarios however where the security decision can not be made on the fly. For these settings we developed a language and a framework for *a-posteriori access control*. In this paper we show how the framework can be used in a *practical scenario*. In particular, we work out the example of an Electronic Health Record (EHR) system, we outline the full architecture needed for audit-based access control and we discuss the requirements and limitations of this approach concerning the underlying infrastructure and its users.

Keywords: distributed access control, audit, accountability, Electronic Health Record (EHR) systems

1 Introduction

Increasingly often, sensitive data needs to be accessed and exchanged across complex and distributed computer systems. To protect data confidentiality, numerous distributed access control mechanisms have been developed. Typically, these systems try to prevent illegitimate actions *before* their occurrence, by deciding on the fly if access should be granted or not.

To see why sometimes a different approach is needed, consider the specific issue of the modernization of health care systems. Roughly, Electronic Health Record (EHR) systems must fulfill two requirements [16]: (1) To provide high-quality health care, the EHR must be immediately available, preferably across the boundaries of the different hospitals and abroad. (2) To protect the patient's privacy, the EHR must remain confidential and should be disclosed only according to the law and/or

* Research funded by SenterNovem through the IOP GenCom program (PAW project).

[1] Email: marnix.dekker@tno.nl

[2] Email: sandro.etalle@utwente.nl

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2006.08.028

the patient's explicit consent. The latter requirement is particularly important in a country such as The Netherlands, where medical insurances are privatized and the patients' medical data is worth millions.

Note that fulfilling both requirements is hard. To fulfill the first requirement, the mechanism should be relatively simple and fast. The second requirement however states that access should only be granted under precise conditions and circumstances. Considering the complexity of the medical work flow, the large number of health records and the variety of institutions, users and systems involved, it seems likely that checking these circumstances and conditions is slow and prone to errors.

Besides the problem of designing a fast access control mechanism that at the same time takes into account complex conditions [1,3,8], it is considered impossible, in general, to design an access control mechanism that models *every* circumstance perfectly [11]; in other words, there are always exceptional, unforeseen circumstances. This is an important issue in the EHR setting, given the mobility of patients and staff, and the urgency of health care. We believe that at least it should be possible for medical staff to self-authorize exceptions to rules, while leaving the process of justifying the exceptions for later.

In this paper we show how the Audit Logic [4,7], an a-posteriori access control framework, can be used in the EHR setting. In the Audit Logic framework, *a-priori* access control is minimized to the mere authentication of users and objects, and their basic authorizations. More complex security procedures are performed in an *a-posteriori* auditing mechanism. Here we formalize a scenario involving medical personnel and health records to show how this access control mechanism works in practice. Finally we discuss the main advantages and drawbacks of using an a-posteriori access control mechanism rather than a preventive one in the specific EHR scenario.

Related work

The Cassandra system [3] was designed to implement an a-priori access control mechanism in an EHR system. The authors test the expressivity of the Cassandra policy language by expressing existing policies regarding access to medical data and activation of medical roles. The policy languages used in the Cassandra system are different flavors of *Datalog with constraints*. While Cassandra's policy language is in theory undecidable, it is argued that this is not problematic in practice. On the other hand, the Audit Logic policy language that we use is not based on Datalog, but on a fragment of first-order logic [5] (which is semi-decidable).

Rissanen et al. [11] address the issue of how to override safely the decisions of a preventive access control system called the Privilege Calculus. At each override a procedure starts to find the appropriate authority which is notified to audit the override. In our approach there is only a minimal preventive access control mechanism, which can not be overriden. Moreover, in our approach it is up to the auditors to decide when and which users to audit.

The policy language used in our framework is most closely related to Binder [1]. In Binder delegation is also modeled with the *says* predicate, which, however, may

not be nested inside another *says* predicate, to keep the language decidable. This restriction is absent in the Audit Logic, and it was shown that the corresponding logic is semi-decidable [5]. Halpern and Weissman [8] also use first-order logic to model security policies. Their setting is centralized and they do not consider a special connective to express delegations.

A related problem is the enforcement of copyrights in content-sharing systems (DRM). It has been argued that DRM can be used to enforce privacy policies. For instance, Conrado et al. [6] propose to use DRM to enable privacy in content-sharing systems and vice versa to use privacy as a driver for a wider use of DRM enabled devices. DRM however, unlike the Audit Logic, requires special (compliant) hardware or software at the application layer. This makes DRM unsuitable for EHR systems or enterprise-privacy systems.

In the context of DRM, a type of a-posteriori access control was proposed by Shmatikov and Talcott [14]. There, a reputation-based trust management (TM) model is presented, in which the reputation of individual agents is determined by the fulfilment or violation of (DRM) licenses. We believe that Trust Management [10], coupled with auditing, may be an interesting solution, especially in large distributed EHR systems.

The justification proofs in the Audit Logic framework are based on a formal logic. A number of distributed access control models are based on formal logics (see the survey by Abadi [1]). In these models an authorization request or an authentication credential corresponds to a logical formula and the authorization or authentication decision corresponds to a proof of the formula. In the PCA framework [2], proposed by Appel and Felten, higher-order logic proofs are generated by the clients, proving that they have the authorization to access webservers. The undecidability of proof finding in higher order logic is not a problem in their setting, because it is the user's task to find the proofs. A similar construction is used in the Audit Logic, where users show auditors that they are accountable by submitting formal proofs, that can be checked automatically.

2 Architecture

In this section we recapitulate the main definitions of the Audit Logic. We refer to earlier work for a more complete description [4].

Policies and Actions

The framework is based on a group of agents executing actions, that can be audited to check whether their actions comply with the relevant (privacy) policies. Actions are modeled by a set of predicates *Act*. The basic set of actions contains $comm(a, b, \phi)$ and $creates(a, d)$: $creates(a, d)$ represents the creation of a piece of data d by agent a, while $comm(a, b, \phi)$ denotes the *communication* of policy ϕ from agent a to agent b. By communicating policies agents delegate rights to other agents to execute certain actions. In addition to the basic actions, there may be scenario-specific actions such as $read(a, d)$, expressing the action of a reading d, or

giveDrug(a, b, c), expressing that a administers b a drug c. Policies are built using a set of atomic predicates, which can be either *permissions*, or *conditions*. The basic permission is $\mathsf{owns}(a, d)$ expressing that a owns d. Additionally, one may have scenario-specific predicates such as $\mathsf{mayRead}(a, d)$, expressing the permission that a may execute the *reads* action on d, or the condition $\mathsf{isNurse}(a)$, expressing that agent a is a nurse. Complex policies are built from permissions and conditions using logical connectives. The grammar of the policy language is [4]:

$$\begin{aligned}\phi ::= {} & \phi \wedge \phi \mid \forall o.\ \phi \mid \phi \rightarrow \phi \mid a \ \mathsf{says}\ \phi \ \mathsf{to}\ b \mid !\alpha \rightarrow \phi \mid ?\alpha \rightarrow \phi \mid \\ ::= {} & \ \mathsf{p}(o_1, ..., o_n) \mid a \ \mathsf{owns}\ d \mid \top.\end{aligned}$$

Here o are objects or variables of the appropriate sort, and $!\alpha \rightarrow \phi$ and $?\alpha \rightarrow \phi$ express *use once* and *use many obligations*, respectively. For example the policy *Each time a nurse gives drug to a patient, agent c can bill that patient* is written as:

$$\big(\mathsf{isNurse}(x) \wedge \mathsf{isPatient}(y)\big) \rightarrow \big(!\mathit{giveDrug}(x, y, c) \ \rightarrow \ \mathsf{mayBill}(c, y)\big)$$

Use-once obligations require special care, in the distributed setting we are considering, to ensure that they are used only to justify a single action [4]. The *owns* predicate models ownership of data, which gives the permission to derive any policy concerning this data, including the permission to delegate permissions about it to other agents. The connectives $\wedge$, $\rightarrow$ and $\forall$ are treated as usual. The *says* construct is a special connective used to express the permission to delegate policies. For example, the policy *Any doctor can delegate to a nurse, the treatment of his patient.* is written as

$$\forall x, y, z.\big(\mathsf{isDoctorOf}(x, y) \wedge \mathsf{isNurse}(z)\big) \rightarrow \ x \ \mathsf{says}\ \mathsf{mayTreat}(z, y)\ \mathsf{to}\ z.$$

Our logic allows to *derive* permissions from a set of actions. We refer to earlier work for a more complete description of the underlying derivation system [5]. Here it is important to be aware of the fact that when an agent a can derive $(b \ \mathsf{says}\ \phi \ \mathsf{to}\ a)$ then a can also derive ϕ.

Logging and Accountability

In the Audit Logic, similarly to what happens in proof-carrying code frameworks [2,17], when an action of an agent is audited, by an *auditor*, the agent has to present a *justification proof* for it. The agent may use (part of) its log as evidence in the justification proof.

For example, consider the two actions executed by agents a and b in Figure 1: In step (1) agent a communicates the policy ϕ to agent b and agent b logs this communication (2), for later. Agent b reads data d (3) and at some moment (4) the Auditing Authority finds this action in some audit trail, in this case the log of queries executed at some database. The auditor decides to ask agent b for justification (5). Agent b justifies, and uses the logged evidence (in step 2) of agent a's policy communication. The auditing authority now asks agent a for a justification of having said ϕ to b.

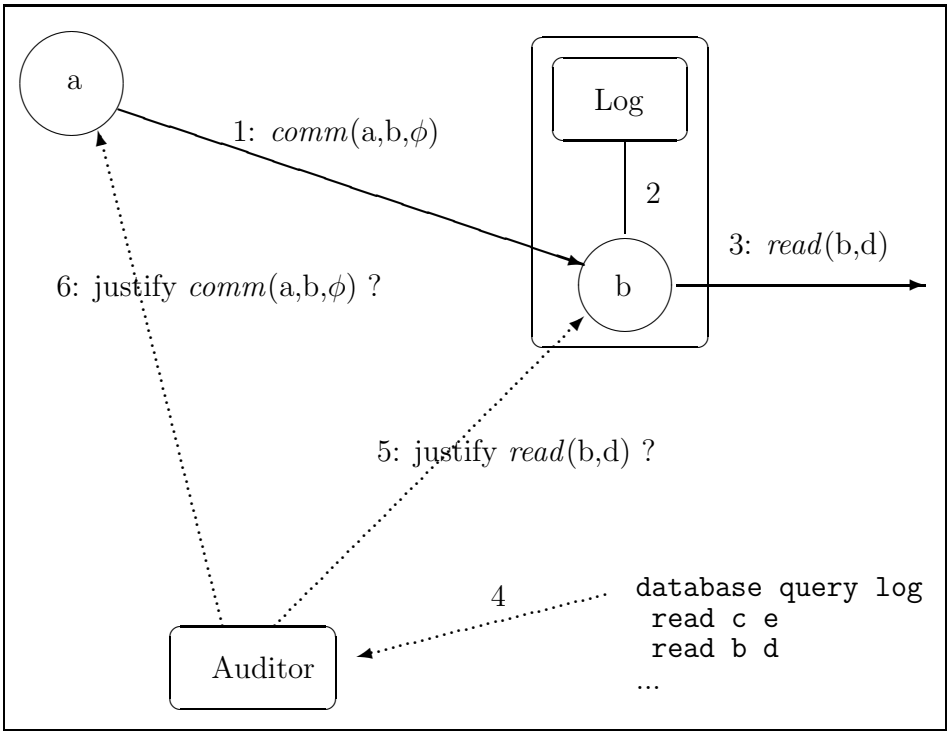

Fig. 1. An overview of the architecture. Agents execute actions and keep logs in order to respond to a possible audit later on. Dotted arrows denote interactions with an auditor. In the picture ϕ denotes the policy $\mathsf{mayRead}(b, d)$.

A formal proof system, denoted $\vdash$, is used to derive justification proofs from evidences in the log. Informally, an agent a can justify an action α from an excerpt ϵ of its log, by proving that

$$\epsilon \vdash_a \phi(\alpha)$$

where ϕ is the proof-obligation for the action α. For example, suppose that the proof-obligation for the action $read(a, d)$ is the formula $\mathsf{mayRead}(a, d)$, then to justify the action $read(a, d)$ to an auditor, agent a has to supply a proof of $\epsilon \vdash_a \mathsf{mayRead}(a, d)$. This proof can then be checked by the auditor. Here we are not interested in specifying how auditors collect evidence and which actions should be audited by the auditors [13]. More details regarding proof-obligations can be found in earlier work [4].

In the event of an audit, an agent needs to *find* justification proofs based on its log. This involves reasoning about possibly complex policies and actions, hence we need a theorem prover. Moreover, a tool is needed for the auditors to *check* the justification proofs; this is an automatic proof checker.

The tool architecture for finding and checking proofs are depicted in Figure 2. For reasons of safety and efficiency both tools are implemented separately using different systems [5]: The proof checker uses Twelf, the proof finder uses SWI Prolog. Shortly, the reason for this distinction is that proof finding is much harder than proof checking. Prolog is a fast application for finding proofs, but the code can be hard to verify for soundness. Proof checking on the other hand is much easier,

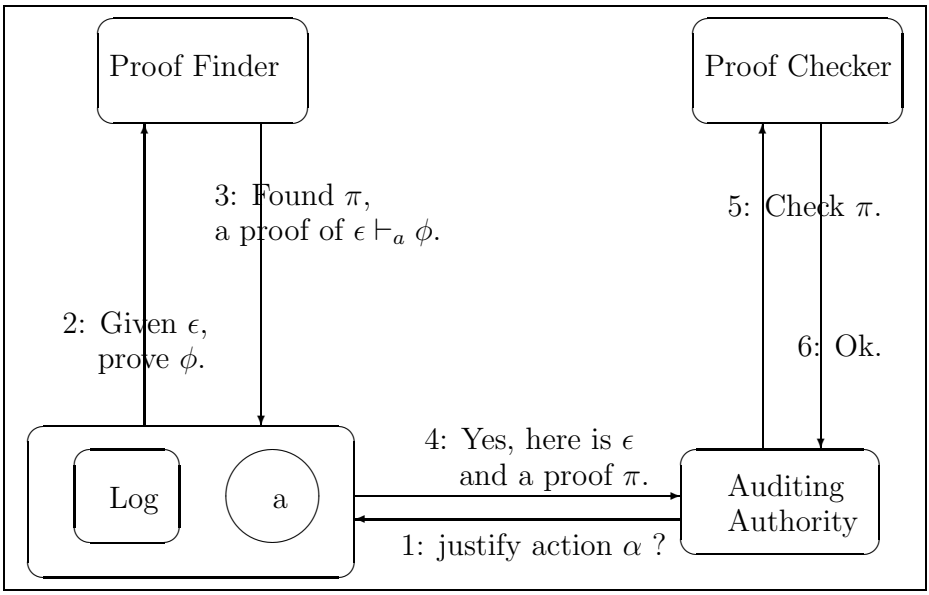

Fig. 2. An overview of the tools for the framework: The proof checker and the proof finder.

and can be done by type-checking. Our type-checker is coded in a few lines of Twelf code which can be checked manually for soundness.
Figure 2 shows how the tools are used in the framework. Assume that an agent a has performed an action α and that the auditing authority wants a to justify α. First, (1) agent a is audited for action α. Agent a now selects an excerpt ϵ of its log and a policy ϕ that is a's proof obligation for action α and (2) tries to find a proof of $\epsilon \vdash_a \phi$ using the proof finder. Then (3) the proof π and the excerpt ϵ are sent to the auditor for checking (4) and finally, (5) the auditor checks that π is indeed a proof of $\epsilon \vdash_a \phi$ by using the proof checker (6). The syntax of the proofs can be seen in Figure 5.

Case Study

To show how the Audit Logic can be used in an EHR setting, we describe a simple scenario involving health records and medical personnel.

Medical records need to be processed and accessed by numerous systems in different places. To protect the patient's privacy and the privacy of medical personnel, users can specify policies that describe who can access the medical record and under which circumstances. In the sequel, we ignore the issues of moving the medical record from one system to the other, instead we focus on the policies that accompany the medical record.

Setup

The agents involved here are patients, doctors, nurses and administrative employees; the users of the EHR system. The data consists of the medical records and the actions we consider are reading a medical record, updating a medical record,

administering medicines and billing a patient for them (cf. Section 2).

The form of the medical records we consider is inspired by the legal directives on privacy of health records [15,16]. A medical record is divided into two parts: first, the personal information, PI, records all non-medical information related to the patient, like billing information, and information regarding the patient's family members (which may have medical records of their own). Second, the medical data (MD) gathers all the medical information of the patient, like diagnoses and given prescriptions. Updates to the medical record are performed by appending new information together with the name of the agent making the update.

Additionally, several auditors have the mandate of controlling that the medical records are used appropriately, i.e. the hospital's internal auditor, a government authority and a patient union representative. Independently, these auditors may audit different sections of the organization.

The hospital has defined a general policy, ϕ_h, to protect the privacy of patients and allows medical personnel to access and handle the necessary health information:

H1 A patient may read and update the PI section of his medical record and authorize others to do so;

H2 A patient may read and update the MD section of his medical record and authorize others to do so;

H3 A doctor may read the PI section of the medical records of his patients;

H4 A doctor may read and update the MD section of the medical record of his patients;

H5 A doctor may give medicines to his patients;

H6 A doctor can delegate to a nurse on his staff the administering of medicines;

H7 An administration employee may bill a patient each time someone has given medicines to that patient;

In Figure 3 the hospital's policy is formalized using the policy grammar from Section 2.

Additional customized policies may be added later to this general policy by the individual users, for example explicit patient consent given to medical staff. Users may send policies to each other by using email.

Note that in this setting whatever is not explicitly mentioned in a policy, is not permitted, and that each policy added entails *more* permissions (monotonicity). A special mechanism would be required to model explicit prohibitions, basically to ensure the proper propagation of these prohibitions in a distributed setting. For an example of such a mechanism we refer to the revocation mechanism of SPKI/SDSI.

2.1 Scenario

Let us instantiate the general setting to a concrete instance. We have patients Alice and Bob, doctors David and Diana, a nurse Natalie and an administrative employee

H1 $\forall a, d.\big(\mathsf{isPatient}(a) \wedge \mathsf{isPI}(a, d)\big) \rightarrow \mathsf{owns}(a, d)$.
H2 $\forall a, d.\big(\mathsf{isPatient}(a) \wedge \mathsf{isMD}(a, d)\big) \rightarrow \mathsf{owns}(a, d)$.
H3 $\forall a, b.\big(\mathsf{isDoctorOf}(b, a) \wedge \mathsf{isPI}(a, d)\big) \rightarrow \mathsf{mayRead}(b, d)$.
H4 $\forall a, b.\big(\mathsf{isDoctorOf}(b, a) \wedge \mathsf{isMD}(a, d)\big) \rightarrow \big(\mathsf{mayRead}(b, d) \wedge \mathsf{mayUpdate}(b, d)\big)$.
H5 $\forall a, b, c.\mathsf{isDoctorOf}(b, a) \rightarrow \mathsf{mayGiveDrug}(b, a, c)$
H6 $\forall a, b, c, d.\big(\mathsf{isDoctorOf}(b, a) \wedge \mathsf{onStaffOf}(c, b)\big) \rightarrow b \text{ says } \mathsf{mayGiveDrug}(c, a, d) \text{ to } c$.
H7 $\forall a, b, c, d.\mathsf{isAdministration}(c) \rightarrow \big(!\mathit{giveDrug}(b, a, d) \rightarrow \mathsf{mayBill}(c, a, d)\big)$.

Fig. 3. The hospital's policy written in the Audit Logic's policy language.

called Charlie. Alice trusts doctor David to give her medical file to other doctors, in case other doctors need to read it, say to get a second opinion. Alice's policy ϕ_a is in words: *Dr. David can delegate the permission to read the MD section of Alice's medical record.* It is formalized as follows:

$$\forall b, d\ \big(\mathsf{isMD}(\mathit{alice}, d) \wedge \mathsf{isDoctor}(b)\big) \rightarrow \mathit{david} \text{ says } \mathsf{mayRead}(b, d) \text{ to } b.$$

Alice's policy, hence, is more specific than the rules in the hospital's policy. The hospital's policy states that only (H4) Alice's doctor may read her medical file. Alice now gives the permission to David, to give read permission also to other doctors. Note that this last

Let us see a sequence of actions performed by the users.

A1 The hospital gives its policy to Dr. David.

A2 Dr. David logs this for later.

A3 Alice becomes patient of Dr. David.

A4 Dr. David logs this for later.

A5 Alice meets Dr. David in his office.

A6 Dr. David reads the PI section of Alice's record, to remind himself of Alice's personal details.

A7 Alice communicates her new policy ϕ_a to Dr. David.

A8 Dr. David logs this communication for later.

A9 Dr. David updates the MD section of Alice's record.

These actions are formalized in the appendix. Note that Dr. David logs the events A1, A3 and A7, because the evidence of these events may be useful for him later on. For example, Dr. David can use A8 later on to prove that he was allowed to show Alice's file to another doctor. The auditors on the other hand may keep an independent (e.g. random) track of actions, in so-called *audit trails*. In particular, the hospital's privacy officer routinely monitors the queries to a database with medical records, both to detect anomalous behavior and to ensure that the hospital's policy is adhered to. The emails between Dr. David and patient Alice, possibly

containing policies are not monitored by the hospital's privacy officer. They may become known to him only because some user uses the policy communication in some justification proof. Independently, an external auditor controls the financial accountability of the hospital, i.e. to ensure that only actual costs are billed to patients.

After some time, the hospital's privacy officer asks Dr. David for a justification for having accessed Alice's file. To give a justification Dr. David needs to show to the auditor the log entries corresponding to A2 and A4 from the sequence above, and a proof that:

$$[A2, A4] \vdash_{david} \mathsf{mayRead}(david, alice).$$

David's proof is automatically checked and the auditor finds that David is accountable.

Now, unexpectedly, Alice arrives injured at the hospital, while Dr, David is off duty. Dr. Diana who is on a shift with nurse Natalie, treats Alice upon arrival:

B1 Dr. Diana logs that Natalie is a nurse on his shift.

Informally Alice asks for treatment.

B2 Dr. Diana reads the MD section of Alice's medical record.

B3 Dr. Diana updates the MD section of Alice's record.

Informally Dr. Diana tells nurse Natalie to give Alice the medicine Qurol.

B4 Natalie administers the medicine.

B5 Natalie notifies billing that Qurol was given to Alice.

B6 Charlie logs this for later.

B7 Charlie bills Alice for the medicine.

B8 Charlie logs this, together with a reference to Natalie's notification.

The actions in this sequence are shown in Figure 4 and formalized in the appendix. Note that both Diana and Natalie operated initially without proper authorizations. Alice's assertion that she is/consents to being Diana's patient was missing. Moreover, Natalie should have had Diana's explicit authorization to administer Qurol.

Say half an hour later, when Alice is a bit better, she authorizes Diana, and, say when the shift is over, Diana authorizes Natalie:

B9 Alice becomes patient of Dr. Diana.

B10 Dr. Diana logs this for later.

B11 Dr. Diana tells nurse Natalie to give Alice the medicine Qurol.

B12 Natalie logs this for later.

Note that the medical staff first treats Alice and then records the necessary details for administration and accountability. Although initially not all the authorizations were available, a-posteriori, the operators can account for their actions. For example, when Alice is asked to account for giving Qurol to Alice, she can send the log

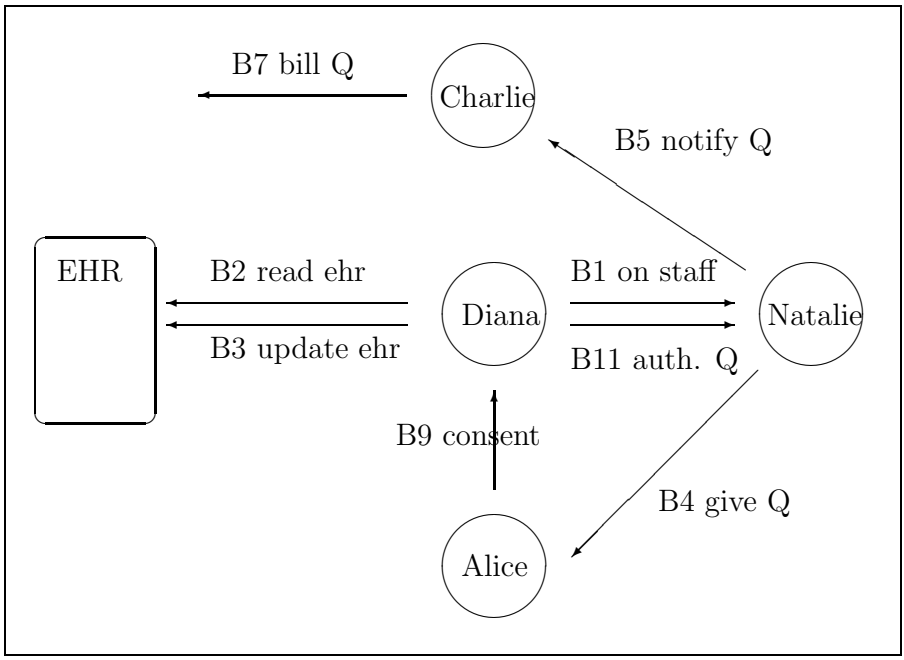

Fig. 4. The sequence of actions involving the patient Alice, where "auth. Q" denotes Dr. Diana's authorization for nurse Natalie to give the medicine Qurol, and "consent" denotes Alice's explicit consent to being treated by Dr. Diana.

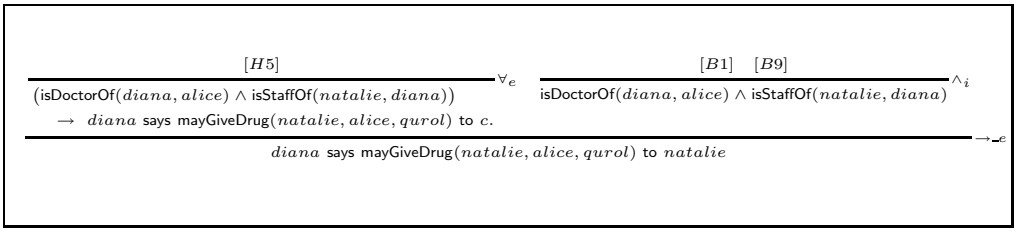

Fig. 5. A sketch of an accountability proof.

entry corresponding to item B11 above, together with a proof of

$$[B12] \vdash_{natalie} \mathsf{mayGiveDrug}(natalie, alice, qurol).$$

To illustrate a more complex proof, Diana's proof for the delegation to Natalie (B11 above) is more involved. To account for it, Diana has to prove that:

$$[H5, B1, B9] \vdash_{Diana} Diana \text{ says } \mathsf{mayGiveDrug}(natalie, alice, qurol) \text{ to } alice.$$

The actual proof involves a number of steps. Diana can use his proof finder to generate the exact proof for him. In Figure 5 we sketch the proof. The auditor can check this proof automatically, using the proof checker. We refer the interested reader to earlier work for more details about the implementation of the tools [5].

Finally, to illustrate the use of *use-once obligations*, the next day Natalie gives another dose of Qurol, in line with what Dr. Diana told her.

C1 Natalie administers the medicine.

C2 Natalie notifies billing that Qurol was given to Alice.

C3 Charlie logs this for later.

C4 Charlie bills Alice for the medicine.

C5 Charlie logs this, together with a reference to Natalie's notification (C3).

Notice that the policy (H7) allowing Charlie to bill Alice for the medicines contains a use-once obligation (cf. Section 2): Each time someone gives drugs to Alice, Charlie can add to Alice's bill. Charlie needs to log the billing action, together with a reference to the corresponding notification by Alice [4]. In other words, when billing Alice, Charlie has to state, in his log, to which notification it belongs. This allows the external auditor, described earlier, to check that each item on the bill corresponds to a dose of Qurol administered by Natalie.

3 Discussion

The case-study shows how a-posteriori access control can be used in the EHR setting. In this section we elaborate on the main advantages and the main drawbacks of this approach.

A-posteriori versus a-priori

The advantage of a-posteriori access control is that it allows the medical staff to go ahead with their duties, without worrying about problems like expiration of certificates, passwords or failing network connectivity to some authorization server. These issues can be dealt with at a more convenient time. By definition a preventive access control system does not provide this kind of flexibility. In practice, an a-priori access control system has to be extended with mechanisms by which users can override the a-priori decisions of the access control mechanism [11]. These overrides must be reviewed later on for legitimacy, and for this purpose users must record the circumstances under which they needed to override. This logging is a central part of the Audit Logic. Useful events or performed actions have to be logged by the medical staff, for the purpose of accountability. By keeping logs of such events and actions, doctors and nurses can account to multiple auditors that come at different times. In a traditional a-priori access control system, on the other hand, the authorizations are only checked once by a single authority, i.e. at the moment access is requested.

A-posteriori access control has a characteristic drawback: it does not prevent misbehavior, hence it does not give the robust security guarantees that are required in e.g. military information systems. In many settings, a-posteriori access control can not replace preventive access control at all because the costs of incidental misuse are much higher than the costs of a (too) preventive security mechanism.

Many countries have adopted legislation that describes rather explicitly the requirements for EHR systems [15,16]. Consider for example the summary of the US act HIPAA [16]. The principal rule is as follows:
A covered entity may not use or disclose protected health information, except either:

(1) as the Privacy Rule permits or requires; or (2) as the individual who is the subject of the information authorizes in writing. [3]

The HIPAA act stresses that patients have the right to justifications of past disclosures of their medical records. In HIPAA it is called *Disclosure Accounting*, being the subject's right to get an accounting of disclosures of its EHR in the past six years.

The a-posteriori access control mechanism, described in this paper, provides a formal definition of this accounting and the tools to automate such accounting. Although not required in the HIPAA act, automation is convenient to be able to run audit tools without human supervision, enhancing both the privacy and the efficiency of the audits.

Finally, about the implementation of the law, the HIPAA act states that *The Privacy Rule does not require that every risk of an incidental use or disclosure of protected health information be eliminated.* Therefore, the main drawback of our approach, i.e. that with a-posteriori access control we can not completely exclude misuse, is in principle allowed for by the law.

Infrastructure

We outline the requirements with respect to the underlying infrastructure.

Actions are executed at the session layer. Here *authentication* and *non-repudiation* is required, to be able to determine which users were responsible for which actions. Once an action is executed, the evidence of its occurrence should be safely stored for later, this is called an *audit trail*. Also, *time-stamping* of actions is required, to be able to demonstrate that an action was not executed before another, or before a required policy was received. Audit trails should provide auditors with a transcript of all (or most) user actions, to detect misbehavior. Consider the example in Figure 1. Here, a database system provides a secure log of past queries to some auditor. On the other hand, the policy communications between the two agents are ignored by the auditor. For example, the users may be using some private email system to communicate (signed) authorizations. The auditors do not monitor these exchanges. Additionally, besides using the logging facilities of electronic systems such as databases and computers, audit trails can also be established using cameras, key-loggers, RFID sensors, database query logs, etc. The audit trails need protection from tampering [13], but this is already required for accounting and security purposes [9].

Moving up to the *application layer*, the main requirement here is the presence of a secure device for users to log the actions and events useful to them. Moreover, both auditors and user may use the tools shown in Section 2 to find and check accountability proofs automatically. These evidences are used by the users in the event of an audit. For example, suppose a doctor changes a medical file and a policy states that in that case the corresponding patient should be notified. To justify his actions later on, he should log the action of notifying the patient. The notification

[3] This was also illustrated in the scenario; patient Alice gave extra permissions to her doctor David.

is executed at the session layer, where it may be caught in an audit trail. At the same time the logging device must create a tuple containing the logged action, its time-stamp and possibly other parameters, which can be used as evidence in an audit later on [4]. To prevent that users forge their logs, some secure device is needed that takes care of this logging.

At the application layer, audit-based access control has little impact, which makes it rather suitable to implement across different institutions, where many different kinds of legacy systems are used. For example, the databases schemas of the involved organizations can remain the same and the ICT infrastructures would not have to be re-designed from scratch.

Privacy and Trust

An important requirement for a-posteriori access control is that there must be some mechanism in place to ensure that users can be held accountable for their actions, i.e. that a user will not vanish after executing his (illegal) actions. This is necessary for the trust of users in each other, and in the EHR system. In real life, this is often done using a bail sum or by some legally binding agreement, such as an employment contract. If this requirement is not met, malicious users can perform actions and disappear before they can be held accountable. In our setting this is particularly important, given the fact that users can give extra permissions to others. A malicious user can set off a cascade of actions by other users.

A more complex aspect is *trust management*. We assumed for simplicity that all users were equally trusted to utter security statements. However, consider the case in which some unknown doctor made changes to the drug prescriptions for Alice. Strictly speaking, Dr. David can trust another doctor, however, Dr. David's employer may require him to make a more complex trust decision that involves checking the foreign doctor's reputation and expertise area. Making such decisions can be supported by using trust management (TM) systems [10]. On the one hand, the TM system should give a full view of how users can be trusted, and on the other hand, when an auditing authority finds that a user is not accountable, it should report this to the TM system to decrease part the user's reputation.

In practice, also in a-priori access control systems some authority checks the system for flaws or abuse by logging and auditing user behavior. In fact this is considered to be essential in conventional access control [12]. Often, these audits are conducted *without* using formal or public procedures. In the auditing approach we have removed the a-priori access control, and provided a formal and automated auditing procedure. Automation allows for fast routine audits, and is more privacy-friendly than auditing by hand. It remains a question if the user's perception of privacy is any different in either approach.

4 Conclusions

In this paper we show how the Audit Logic [4,7] can be used in the setting of Electronic Health Records. We outlined the full architecture and we discussed the

advantages and drawbacks of this approach, regarding ICT infrastructure and the users' privacy and trust.

We showed that our approach requires minimal changes to the infrastructure. At the lower layers, secure audit trails, with all (or most) of the user actions, are already required [9]. By minimizing the a-priori access control and relying on a-posteriori access control we get a more flexible system to adapt to the medical information flow. Yet the audit mechanism provides the necessary assurance, later on, that the staff complied to the relevant policies. A-posteriori access control is convenient when authorizations are not available on the moment that access to medical files is needed. Moreover, as mentioned before, this type of auditing is required by legislation concerning EHR systems [16]. In the Audit Logic framework such audits are performed by a formal and automated auditing procedure.

In the EHR setting, privacy is an important issue for both patients and medical staff. A-posteriori access control is in theory more intrusive to the user's privacy than a-priori access control. A-priori access control however is also coupled with audits of logs and user actions [12,13]. An automatic audit procedure not only enhances the privacy of the patients, but also that of the medical staff, wary perhaps of human auditors that go through the logs by hand. Formal and public auditing procedures make the privacy protection mechanism also more transparent to the patients as well as to the medical staff. In the future we wish to investigate how we can control the actions of the auditors in turn and how we can achieve maximal privacy of the (honest) users with respect the auditors.

Acknowledgement

We would like to thank Ricardo Corin and Pieter Hartel for helpful discussions about this paper. We also thank the anonymous reviewers for their comments and suggestions.

References

[1] Abadi, M., *Logic in access control*, in: P. G. Kolaitis, editor, *Proc. of the 18th Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society Press, 2003, pp. 228–233.

[2] Appel, A. W. and E. W. Felten, *Proof-carrying authentication*, in: G. Tsudik, editor, *Proc. of the 6th Conference on Computer and Communications Security (CCS)*, ACM Press, 1999, pp. 52–62.

[3] Becker, M. Y. and P. Sewell, *Cassandra: Flexible trust management, applied to electronic health records*, in: R. Focardi, editor, *Proc. of the 17th Computer Security Foundations Workshop (CSFW)*, IEEE Computer Society Press, 2004, pp. 139–154.

[4] Cederquist, J. G., R. Corin, M. A. C. Dekker, S. Etalle and J. I. den Hartog, *An audit logic for accountability*, in: W. Winsborough and A. Sahai, editors, *Proc. of the 6th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, IEEE Computer Society Press, 2005, pp. 34–43.

[5] Cederquist, J. G., R. Corin, M. A. C. Dekker, S. Etalle, J. I. den Hartog and G. Lenzini, *The Audit Logic - Policy Compliance in Distributed Systems*, Technical Report TR-CTIT-06-33, 2006.

[6] Conrado, C., M. Petkovic, M. van der Veen and W. van der Velde, *Controlled sharing of personal content using digital rights management*, in: E. Fernández-Medina, editor, *Proc. of the 4th International Workshop on Security In Information Systems (WOSIS)*, 2005, pp. 173–185.

[7] Corin, R., S. Etalle, J. I. den Hartog, G. Lenzini and I. Staicu, *A logic for auditing accountability in decentralized systems*, in: T. Dimitrakos and F. Martinelli, editors, *Proc. of the 2nd IFIP Workshop on Formal Aspects in Security and Trust (FAST)*, volume **173**, Springer, 2004 pp. 187–202.

[8] Halpern, J. Y. and V. Weissman, *Using first-order logic to reason about policies*, in: R. Focardi, editor, *Proc. of the 16th Computer Security Foundations Workshop (CSFW)*, IEEE Computer Society Press, 2003, pp. 187–201.

[9] Jajodia, S., S. Gadia and G. Barghava, *Information Security: An Integrated Collection of Essays*, chapter Logical design of audit information in relational databases, IEEE Computer Society Press, 1995.

[10] Li, N., J. Mitchell and W. Winsborough, *Design of a role-based trust-management framework*, in: M. Abadi and S. M. Bellovin, editors, *Proc. of the Symposium on Research in Security and Privacy (S&P)*, IEEE Computer Society Press, 2002, pp. 114-130.

[11] Rissanen, E., B. Sadighi Firozabadi and M. J. Sergot, *Discretionary overriding of access control in the privilege calculus*, in: T. Dimitrakos and F. Martinelli, editors, *Proc. of the 2nd IFIP Workshop on Formal Aspects in Security and Trust (FAST)*, Springer, 2004, pp. 219–232.

[12] Sandhu, R. and P. Samarati, *Access control: Principles and practice*, IEEE Communications Magazine **32(9)** (1994), pp. 40–48.

[13] Sandhu, R. and P. Samarati, *Authentication, access control, and audit*, ACM Computing Survey **28(1)** (1996), pp. 241–243.

[14] Shmatikov, V. and C. L. Talcott, *Reputation-based trust management*, Journal of Computer Security **13(1)** (2005), pp. 167–190.

[15] The European Parliament and the Council of the European Union, UE DIRECTIVE 2002/58/EC on privacy and electronic communications.

[16] The US Dpt. of Health and Human Services, Summary of the HIPAA Privacy Rule.

[17] Whitehead, N., M. Abadi and G. C. Necula, *By reason and authority: A system for authorization of proof-carrying code*, in: R. Focardi, editor, *Proc. of the 17th Computer Security Foundations Workshop (CSFW)*, IEEE Computer Society Press, 2004, pp. 236–250.

A Appendix

For the sake of brevity, the two sequences of actions depicted in Section 2 are formalized here.

The first sequence, where patient Alice visits Dr. David in his office:

$$\begin{aligned}
&\text{A1 } comm(hospital, david, \phi_h).\\
&\text{A3 } comm(alice, david, \mathsf{isDoctorOf}(david, alice)).\\
&\text{A6 } read(david, md_alice).\\
&\text{A7 } comm(alice, david, \phi_a).\\
&\text{A9 } update(david, md_alice).
\end{aligned}$$

The second sequence, where Alice arrives, unexpectedly, in the hospital, while

Dr. Diana is on duty:

B1 $comm(natalie, diana, \mathsf{isOnStaffOf}(natalie, diana))$.
B2 $read(diana, md\_alice)$.
B3 $update(diana, md\_alice)$.
B4 $giveDrug(natalie, alice, qurol)$.
B5 $notify(natalie, charlie, qurol, alice)$.
B7 $bill(charlie, alice, qurol)$.
B9 $comm(alice, diana, \mathsf{isDoctorOf}(diana, alice))$.
B11 $comm(diana, natalie, \mathsf{mayGiveDrug}(natalie, alice, qurol))$.