A Logic for Auditing Accountability in Decentralized Systems

R. Corin[1], S. Etalle[1,2], J. den Hartog[1], G. Lenzini[1], and I. Staicu[1]

[1]Dep. of Computer Science, University of Twente, The Netherlands
[2] CWI, Center for Mathematics and Computer Science Amsterdam
{corin,etalle,hartogji,lenzinig,staicui}@cs.utwente.nl

Abstract. We propose a language that allows agents to distribute data with usage policies in a decentralized architecture. In our framework, the compliance with usage policies is not enforced. However, agents may be audited by an authority at an arbitrary moment in time. We design a logic that allows audited agents to prove their actions, and to prove their authorization to posses particular data. Accountability is defined in several flavors, including *agent* accountability and *data* accountability. Finally, we show the soundness of the logic.

1 Introduction

Consider the following scenario: Alice gives marketing company BigBrother some personal information (e.g., her spending patterns, music preferences, part of her medical record, ...), in exchange for some bonus miles. In addition, Alice allows BigBrother to sell to a third party part of this information, but only if anonimized and under provision that she will receive 10 percent of the revenues.

The problem here is, how can we make sure that the data is being used only according to Alice's wishes. Notice that in the above scenario BigBrother might sell Alice's data to BigSister, who in turn might sell part of it to SmallNephew, and so on.

This problem is not only that of privacy protection in a distributed setting. In fact, modern scenarios of digital asset delivery (where a digital asset can be anything ranging from a piece of private information to a movie or a character in a multiplayer game) are departing from the usual schemas in which the assets are equipped with an immutable usage policy that applies to the whole distribution chain. Instead, we are moving towards a situation in which information brokers collect, combine and redistribute digital assets.

The question that needs to be answered here is how can we describe and enforce usage policies in such a decentralized dynamic, evolving context.

In this paper we present a logic data access and agent accountability in a setting in which data can be created, distributed and re-distributed. Using this logic, the owner of the data attaches a usage policy to the data, which contains a logical specification of what actions are allowed with the data, and under which conditions. This logic allows for different kind of accoutability and it is shown to be sound.

Part of problem we are tackling is that of enforcing that agents actually follow the behaviour that policies dictate; in general, this is a difficult task, typically requiring continuous monitoring of agents, which is usually infeasible. Therefore, we consider an alternative to policy enforcement, based on an analogy with the real world, where people are not always controlled for correct behaviour. Instead, eventually an agent (say Alice) might be *suspected* of incorrect behaviour; in that case, an authority would *query* Alice for a justification of her actions. This justification can be supported by *evidence*, that the authority can check and validate.

2 System, Syntax and running example

Our system consists of a group of communicating agents which create and share data and an authorization authority which may audit agents. The creation of data, as well as the communcation between agents, is assumed to leave some evidence and hence is observable (from the perspective of the authorization authority). As we do not continuously monitor agents, the internal computations of agents are not considered to be observable. However, when auditing an agent, the data and policies currently stored by an agent become visible to the authorization authority. Thus, the model of an agent consists of storage, (unobservable) internal computation and (observable) actions such as communication.

Example 1. As a *running example* we consider a scenario with three agents, a content provider Alice (a), a reviewer/distributer Bob (b) and a user Charlie (c). In this setting content provider Alice creates content d and sends it to Bob for review with permission for Bob to read the data but not to retransmit it, in effect protecting the data with a non-disclosure agreement (NDA). After some time Alice lifts the NDA by giving Bob permission to resend the data to Charlie.

Bob sends the data on to Charlie with permission to read the data. Charlie does not produce any observable actions but the policy allowing him to read d is in his storage after Bob sends this policy.

The following subsections introduce the logical language used to express policies and describe the system in more detail.

2.1 The syntax

For the formal model we will use a set of *agents* $\mathcal{G}$ ranged over by a, b and c and a set of *data objects* $\mathcal{D}$ ranged over by d. As the order of actions can be relevant we also introduce a notion of (global discrete) time described by using a well-founded totally ordered set $\mathcal{T}$, ranged over by t. (In examples we will use the natural numbers for $\mathcal{T}$.)

The *policy formulae*, expressing data usage policies, further require a set of *predicates* $\mathcal{C}$, ranged over by p, which express basic operations that can be performed on data. For example, $\mathsf{read}(a, d)$ and $\mathsf{print}(a, d)$ respectively indicate that user a may read and print data d. For readability we will restrict our definitions to binary predicates taking a single agent and a single data object.

Definition 2. *The set of* policy *formulae Φ, ranged over by ϕ and ψ, is defined by the following grammar:*

$$\phi ::= p(a, d) \mid a \ \mathsf{owns}\ d \mid a \ \mathsf{says}\ \phi \ \mathsf{to}\ b \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi$$

with $a, b \in \mathcal{G}$, $d \in \mathcal{D}$, $p \in \mathcal{C}$.

First, a policy formula can be a simple predicate $p(a, d)$, such as $\mathsf{read}(a, d)$ mentioned above. Second, we have the a owns d formula. This formula indicates that a is the owner of data object d. As we will see below, an owner of data can create usage policies for that data. A third construction is a says ϕ to b which expresses the claim that agent a is allowed to give policy ϕ to agent b. The 'says' contains a target agent to which the statement is said instead of the broadcast interpretation used for a similar construct in e.g. [10, 1]. This allows us to provide a precise way of expressing policies to certain agents. Finally, the logic constructions *and*, *or* and *implication* have their usual meaning.

The *base data set* of a policy formula ϕ, denoted $dv(\phi)$, consists of the data objects the policy refers to. It is defined as one whould expect:

$$dv(a \ \mathsf{owns}\ d) = dv(p(a, d)) := \{d\}$$
$$dv(a \ \mathsf{says}\ \phi \ \mathsf{to}\ b) := dv(\phi)$$
$$dv(\phi \wedge \psi) = dv(\phi \vee \psi) = dv(\phi \rightarrow \psi) := dv(\phi) \cup dv(\psi)$$

We denote a formula ϕ whose base data set is D, i.e. $dv(\phi) = D$, as $\phi[D]$. If D is a singleton set, i.e. $D = \{d\}$, we simply write $\phi[d]$. Note that the base data set of a formula is always non-empty.

Example 3. The policy which allows Bob to read the data d is expressed as $\mathsf{read}(b, d)$. Allowing agent Bob to send to data on to Charlie provided he already has permission to read it is expressed by $\mathsf{read}(b, d) \rightarrow b$ says $\mathsf{read}(c, d)$ to c.

Observables Beside usage policies we also have the observable actions of agents. We will use *evidence formulae* to describe observable actions. As mentioned above, communication and creation of data are the observable actions possible in our system. For simplicity we will only consider these two types of observable actions though extension with other types of observable actions is possible.

Definition 4. *The set of* evidence formulae EV*, ranged over by* ev*, is defined by the grammar:*

$$\mathsf{ev} ::= \mathit{creates}(t, a, d) \mid \mathit{comm}(t, a \Rightarrow b, \phi)$$

with $t \in \mathcal{T}$ $a, b \in \mathcal{G}$*,* $d \in \mathcal{D}$ *and* $\phi \in \Phi$*.*

First, we have the $\mathsf{creates}(t, a, d)$ evidence formula which states that an agent a has created a piece of data d at time t. As we shall see later, this will automatically make a the owner of d. Secondly, we have a communication evidence formula $\mathsf{comm}(t, a \Rightarrow b, \phi)$, which states that agent b has received a policy formula ϕ from agent a at time t. To refer to the time of an evidence formula we define the function $\mathsf{time} : EV \to \mathcal{T}$ as $\mathsf{time}(\mathsf{creates}(t, a, d)) = \mathsf{time}(\mathsf{comm}(t, a \Rightarrow b, \phi)) := t$.

Example 5. The formula $\mathsf{creates}(0, a, d)$ expresses that Alice created data d at time 0. The formula $\mathit{comm}(1, a \Rightarrow b, \mathsf{read}(b, d))$ expresses that Alice sent the permision for Bob to read d to Bob at time 1.

2.2 The model

The observable actions executed by agents (in a run of the system) are combined in the so called *evidence set* $\mathcal{E}$. We will simply use evidence formulae to describe these observable actions, i.e. $\mathcal{E} \subseteq EV$. We make the natural restriction that only finitely many actions can be executed at any given moment in time.

As mentioned before, agents may be audited by an authorization authority, say at time T. At this time each agent $a \in \mathcal{G}$ has a state $\mathcal{S}_a$, which represents a's storage, where all her present data policies are stored.

Example 6. We can now complete the formal description of our running example. The actions of the agents are shown in Figure 1.

$$\begin{aligned}\mathcal{E} = \{\ & \mathit{create}(0, a, d), \mathit{comm}(1, a \Rightarrow b, \mathsf{read}(b, d)), \mathit{comm}(2, b \Rightarrow c, \mathsf{read}(c, d)), \\ & \mathit{comm}(3, a \Rightarrow b, \mathsf{read}(b, d) \to b\ \mathsf{says}(\mathsf{read}(c, d))\ \mathsf{to}\ c), \mathit{comm}(4, b \Rightarrow c, \mathsf{read}(c, d))\}\end{aligned}$$

Note that Bob actually sends the read permission for the data to Charlie twice. The second time this is fine but the first time violates the NDA.

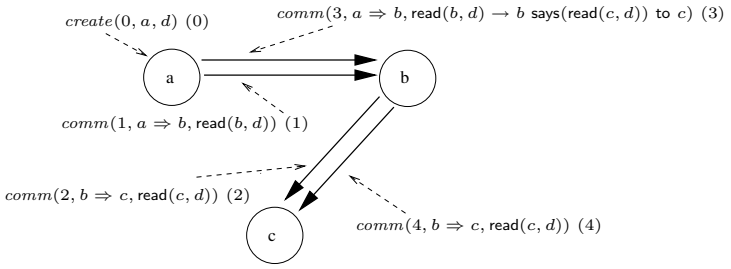

Fig. 1. Solid arrowed lines represent communications, dashed lines represent observable events.

This gives the following evidence set:

When charlie is audited at time 5 his storage $\mathcal{S}_c$ contains (among others) $\mathsf{read}(c, d)$. Note that this is discovered by the authority examining his storage, it does not follow, e.g. from the communication or other observable actions. Appearently charlie has done some (unobservable) internal computation to arrive at the conclusion that he may read d. The question now is, did Charlie correctly conclude that he was allowed to read d and has anything unauthorized happened to the data. We will address this issue in the next section.

3 Using usage policies: The proof system

This section describes the proof system used to derive the actions on data that are allowed by the policies that a user posseses. We first give the inference rules followed by the notion provable. Note that each agents locally reasons about policies therefore the rules include the *subject*, i.e. the agent doing the reasoning. Inference rules have the following format:

$$\texttt{NAME}\ \frac{\mathit{premises}}{\mathit{conclusions}}\ \mathit{subject}$$

where a *premise* can be either a policy formula (in Φ) or an evidence formula (in EV), *conclusion* is a policy formula, and *subject* is an agent in $\mathcal{G}$.

The rules of our proof system are presented in two parts. First, we have the standard rules from the propositional logic

$$\wedge\mathtt{I}\ \frac{\phi \quad \psi}{\phi \wedge \psi}\ a \qquad \wedge\mathtt{EL}\ \frac{\phi \wedge \psi}{\phi}\ a \qquad \wedge\mathtt{ER}\ \frac{\phi \wedge \psi}{\psi}\ a \tag{1}$$

$$\vee\mathtt{IL}\ \frac{\phi}{\phi \vee \psi}\ a \qquad \vee\mathtt{IR}\ \frac{\phi}{\psi \vee \phi}\ a \qquad \vee\mathtt{E}\ \frac{\phi \vee \phi' \quad \overset{[\phi]}{\psi} \quad \overset{[\phi']}{\psi}}{\psi}\ a \tag{2}$$

$$\mathtt{MP}\ \frac{\phi \to \psi \qquad \phi}{\psi}\ a \qquad\qquad \to\mathtt{I}\ \frac{\overset{[\psi]}{\phi}}{\psi \to \phi}\ a \tag{3}$$

We have the standard rules for *and* introduction and elimination in row (1), *or* introduction and elimination in row (2) and *implication* introduction and elimination (modus ponens) in row (3). The notation $\overset{[\phi]}{\psi}$ represents that the proof of ψ has ϕ as a *temporary assumption*.

The second part of proof system consist of the following rules which deal with creation of policies and with the delegation of responsibility (COMM). Note that these rules do not take time or existence of evidence into account. This will be done in our notion of (authorization) proofs below.

$$\mathtt{COMM}\ \frac{\mathsf{comm}(t, a \Rightarrow b, \phi)}{a\ \mathsf{says}\ \phi\ \mathsf{to}\ b}\ b \tag{4}$$

$$\mathtt{CREATES}\ \frac{\mathsf{creates}(t, a, d)}{a\ \mathsf{owns}\ d}\ a \tag{5}$$

$$\mathtt{SAY}\ \frac{a\ \mathsf{says}\ \phi\ \mathsf{to}\ b}{\phi}\ b \tag{6}$$

$$\mathtt{DERPOL}\ \frac{a\ \mathsf{owns}\ d_1 \quad \ldots \quad a\ \mathsf{owns}\ d_n}{\phi[\{d_1, \ldots, d_n\}]}\ a \tag{7}$$

Rule (COMM) states that if agent b has recieved message ϕ from an agent a at some time, then b may conclude the corresponding $a\ \mathsf{says}\ \phi\ \mathsf{to}\ b$ formula. Rule (CREATES) expresses that by creating a piece of data, the agent becomes the owner of that data. Rule (SAY) expresses deligation of responsibility. If agent a says ϕ to b then b can assume ϕ to hold. It is a's responsibility to show that it had permission to give ϕ to b. Note that in our current setup it would also have been possible to omit this rule and derive ϕ directly in rule (COMM). We expect, however, that

with extension of our logic the seperation of these two steps will become useful. Rule (DERPOL) allows the creation of policies. An agent a can create any usage policy for data that she owns.

3.1 Building Proofs

We are ready to introduce proofs built from our logic system. The first definition states what is in fact a proof for some agent x.

Definition 7. *A* proof $\mathcal{P}$ *of* ϕ *for* x *is a finite derivation tree such that:*

1. *each rule of* $\mathcal{P}$ *has* x *as its subject;*
2. *each rule of* $\mathcal{P}$ *belongs to one of the above rules* (1)-(7)*;*
3. *the root of* $\mathcal{P}$ *is* ϕ*;*

Given a proof $\mathcal{P}$, we write $prem(\mathcal{P})$ for the set of premises in the initial rules of $\mathcal{P}$ which are *not* temporary assumptions (like in rules $\vee E$ and $\rightarrow I$). We also write $conc(\mathcal{P})$ to denote the conclusion of the last rule of $\mathcal{P}$, and $subject(\mathcal{P})$ to denote the subject.

For auditing purposes, we want to restrict to proofs that only have evidence formulae as premises and whose time of the evidences is bounded. We call such proofs *justification* proofs.

Definition 8. *A proof* $\mathcal{P}$ *(of* ϕ *for* x*) is called a* justification proof *(of* ϕ *for* x*)* at time t *if every formula in* $prem(\mathcal{P})$ *is an evidance formula* ev *satisfying* $time(ev) < t$.

The set of all justification proofs is denoted by $\mathcal{J}$.

Note that a justification proof at time t is just a proof which is potentially valid at time t. Any evidance formula can be used as a premise. To check whether the proof is indeed valid, a link has to be made with the actions observed, i.e. those in the the set $\mathcal{E}$. This will be done in the next section. It is easy to see that justification proofs are *monotonic*, i.e. any proof that is a justification at time t is also a justification at time t' for any $t' > t$.

As an aside, our policy language is negation free and all proofs have a 'constructive' flavoour. For extensions of the logic, it may be neccessary to go to intuitionistic or linear logic altogether. The constructive nature of the proofs inherently means that the derivation system is not complete: For example, $\mathsf{read}(c, d) \rightarrow \mathsf{print}(c, d)$ can hold simply because $\mathsf{read}(c, d)$ does not. However, there is no constructive derivation for this (Also, as soon as read permission is obtained, the predicate may no longer hold.)

Example 9. In our running example agent Charlie can provide an justification proof for $\mathsf{read}(c, d)$ at time 5 as follows.

$$\mathcal{P}_1 \left\{ \begin{array}{l} \texttt{COMM}\ \dfrac{\mathsf{comm}(4, b \Rightarrow c, \mathsf{read}(c, d))}{\qquad\qquad\qquad\qquad}\ c \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \\ \texttt{SAY}\ \dfrac{b\ \mathsf{says}\ \mathsf{read}(c, d)\ \mathsf{to}\ c}{\mathsf{read}(c, d)}\ c \end{array} \right.$$

Note that replacing the first premise by $\mathsf{comm}(0, b \Rightarrow c, \mathsf{read}(c, d))$ also gives a justification proof for $\mathsf{read}(c, d)$ at time 5. This second proof should not be accepted by the authorization authority as Bob did not actually send anything to Charlie at time 0. The next section will treat what agents should proof and which proofs are accepted by the authority.

4 Accountability

As noticed in the example in the previous section, agents can potentially provide different justification proofs. We model an agent *providing* a proof of ϕ at time t as a function $\texttt{Pr} : \Phi \times \mathcal{G} \times \mathcal{T} \to \mathcal{J} \cup \{\bot\}$. Here the value $\bot$ represents that the agent cannot provide a proof.

We present two notions of accountability. The first notion, *agent* accountability, focuses on whether the actions of a given agent where authorized. The second notion, *data* accountability, expresses that a given piece of data was not misused.

Recall that in our system, an agent a can be audited at time T at which point $\mathcal{S}_a$, the storage of a, becomes visible to the authorization authority. The observable actions performed in the system are collected in $\mathcal{E}$. For both notions of accountability, it is important to link proofs to actual observable actions. To this end we introduce the notion of *authorization proof*, which is a justification proof that is backed by actual evidence.

Definition 10. *We say that a justification proof $\mathcal{P}$ of ϕ for a at time t is* authorized*, written $\texttt{Aut}(\mathcal{P})$, when $prem(\mathcal{P}) \subseteq \mathcal{E}$. In this case we call $\mathcal{P}$ an* authorization proof *of ϕ for a at time t.*

An agent is accountable for the the policies she posseses *and* for the usage policies she gives to others. Thus to pass the audit, the agent needs to authorize her storage and her communication.

Definition 11 (Accountability of a). *We say that agent a is authorised to have ϕ at time t, denoted $\texttt{Aut}_\phi(a, t)$, if she provides an authorization proof, i.e. $\texttt{Pr}(\phi, a, t) \neq \bot$ and $\texttt{Aut}(\texttt{Pr}(\phi, a, t))$.*

We write $\texttt{Aut}(a)$ if a is authorized to have all usage policies in her storage at the time T of auditing, i.e. $\forall \phi \in \mathcal{S}_a : \texttt{Aut}_\phi(a, T)$

We write $\texttt{ComAut}(a)$ *if a was authorized to send all the policies that she did send, i.e.* $\forall \mathsf{comm}(t, a \Rightarrow b, \psi) \in \mathcal{E} : \texttt{Aut}_{a \text{ says } \psi \text{ to } b}(a, t)$.

Finally, we say that agent a passes the accountability *test, written* $\texttt{Acc}(a)$*, if both* $\texttt{Aut}(a)$ *and* $\texttt{ComAut}(a)$*. hold.*

Example 12. In our running example Charlie can show to be authorized for having $\mathsf{read}(c, d)$ by providing the proof from Example 9. Assuming he is also authorized for other policies in his storage we have $\texttt{Aut}(c)$ and also $\texttt{Acc}(c)$ as Charlie did not send any messages (so $\texttt{ComAut}(c)$ is emptily satisfied).

Bob, on the other hand, cannot pass the accountability test as he cannot provide an authorization for b says $\mathsf{read}(c, d) \rightarrow c$ at time 2.

Agent accountability is useful to check the behaviour of a single agent. However, a data owner may be more interested in whether a specific piece of data (with corresponding usage policy) was obtained correctly. To describe this we introduce the notion of data accountability.

4.1 Data Accountability

Data accountability describes the authorization requirements for a single data usage policy. Unlike agent accountability, this may require authorizations from several different agents. We first introduce weak data accountability, which describes that a given usage policy *may* have been obtained correctly. We will then discuss some potential issues with this notion and introduce the notion of strong data accountability.

Weak data accountability expresses that an agent must provide a authorization proof and that all delegated responsibilities must also be accounted for, i.e. for any recieved policies used to derive the policy, there is data accountability for sending of that policy at the sending agent.

Definition 13 (Weak Data Accountability). *We say that* ϕ *at* a *passes the* weak data accountability *test at time* t*, written* $\texttt{Dac}(\phi, a, t)$ *if* a *is authorised to have* ϕ *at time* t *(i.e.* a *provides an authorization proof) and for all communications in the premise of the provided authorization proof,* $\mathsf{comm}(t', b \Rightarrow a, \psi) \in prem(\texttt{Pr}(\phi, a, t))$*, we have* $\texttt{Dac}()b$ *says* ψ *to* a, b, t'*.*

We write $\texttt{Dac}()\phi, a$ *for weak data accountability at the time* T *of the audit, i.e. for* $\texttt{Dac}()\phi, a, T$*.*

Note that this recursive definition is unproblematic, as time must decrease ($t' < t$) by definition of authorization proof and time is well founded. Weak data accountability corresponds to either of the proofs

depicted in solid or dashed lines (but not both) derivations in Figure 2-(C). Intuitively, after checking authorization of ϕ, we 'recurse' to the sending agents where data accountability is checked for the policy which allowed sending the communication.

If data accountability does not hold, then we can deduce that, at some point, some agent did not provide an authorization proof. Clearly this agent does not pass the agent accountability test. The proof of the following proposition is straightforward.

Proposition 14. *If* $\mathtt{Dac}()\phi$ *does not hold, then* $\exists a \in \mathcal{G}$ *such that* $\mathtt{Acc}(a)$ *does not hold.*

Example 15. Weak data accountability of $\mathsf{read}(c, d)$ at c implies that Charlie needs to provide an authorization proof. If Charlie provides the proof given in example 9 then data accountability of b says $\mathsf{read}(c, d)$ to c for Bob at time 4 will be required. Bob can indeed provide an authorization proof:

$\mathsf{read}(b, d) \rightarrow (b \text{ says } (\mathsf{read}(c, d)) \text{ to } c)$.

$$\mathtt{MP}\ \dfrac{\mathcal{P}_2 \left\{ \mathtt{COMM}\ \dfrac{\mathsf{comm}(1, a \Rightarrow b, \mathsf{read}(b, d))}{\mathtt{SAY}\ \dfrac{a \text{ says } \mathsf{read}(b, d) \text{ to } b}{\mathsf{read}(b, d)}\ b}\ b \right. \quad \mathcal{P}_3 \left\{ \mathtt{COMM}\ \dfrac{\mathsf{comm}(3, a \Rightarrow b, \psi)}{\mathtt{SAY}\ \dfrac{a \text{ says } \psi \text{ to } b}{\psi}\ b}\ b \right.}{b \text{ says } \mathsf{read}(c, d) \text{ to } c}\ b \tag{8}$$

Clearly Alice can provide authorization proofs for the two policies she sent as she, being the owner of the data, may create any policy. Thus we have $\mathtt{Dac}(\mathsf{read}(c, d), c, 5)$.

We do not have $\mathtt{Dac}(\mathsf{read}(c, d), c, 3)$. The only authorization proof Charlie can provide uses the fact that Bob sent read permission at time 2. As we have seen before, Bob cannot authorize sending this permission at time 2.

The example above shows an issue with weak data accountability. The result of the data accountability check depends on the authorization proof that Charlie provides. Both the proof using Bobs read permission at time 2 and at time 4 could be used by Charlie. If Charlie and Bob are working together to try to hide that Bob did something wrong, the weak data accountability test of $\mathsf{read}(c, d)$ for Charlie at time 5 will not reviel that Bob violated the NDA.

To capture situations like this we introduce the notion of strong data accountability. As the internal computations of an agent are not visible, the authority cannot check if the provided proof is the proof an agent actually used to arrive at a policy. Or even if the agent created a correct

proof at all before using the policy. The fact that there is no way to check this is an unpreventable limitation due to the unobservability of some of the agents actions. We can, however, check all correct proofs an agent could have used to obtain a policy. This will allow us to prevent situation as in the example above where Charlie behaves correctly but can still hide Bobs violation of the NDA. With *strong data accountability* we do not look at the authorization proof the agent provides but instead look at all (reasonable) proofs. In this way we force checking of all communication that may have been used to derive a policy.

A *minimal proof* $\mathcal{P}$ of ϕ is a proof of ϕ for which there are no unnecessary premises, i.e. there is no proof of ϕ using a strict subset of $prem\mathcal{P}$ as premises.

Definition 16 (Strong Data Accountability). *We say that ϕ at a passes the* strong data accountability *test at time t, written $\mathtt{SDac}(\phi, a, t)$ if a is authorised to have ϕ at time t and for all minimal authorization proofs $\mathcal{P}$ of ϕ for a at time t and all $\mathsf{comm}(t', b \Rightarrow a, \psi)$ in $prem(\mathcal{P})$, we have $\mathtt{SDac}(b \ \mathsf{says}\ \psi \ \mathsf{to}\ a, b, t')$.*

We write $\mathtt{SDac}(\phi, a)$ for strong data accountability at the time T of the audit, i.e. for $\mathtt{SDac}(\phi, a, T)$.

Strong data accountability corresponds to following both the solid or dashed lines in Figure 2-(C). If we assume that agents provide minimal proofs, strong data accountability is a stonger notion that weak data accountability. Checking strong data accountability, however, requires a much more capable authorization authority. For weak data accountability, the agents are required to provide proofs. In this case, it is in the agents interest to show that the communications used in the proof have indeed happened. Thus a setup which uses undeniable communications, e.g. through use of some non-repudian scheme, will be sufficient. For strong data accountability, the authority needs to find and check all relevent communications looking e.g. at communication logs and/or using key escrow techniques.

Example 17. We do not have strong data accountability of $\mathsf{read}(c, d)$ for Charlie at time 5. Altough Charlie can provide authorization, checking all possible minimal proofs will also lead to checking the communication from Bob to Charlie at time 2 which Bob cannot authorize.

5 Semantics

Even though the meaning of our logic operators is intuitive, in this section we shall make that more precise and define a semantic evaluation function

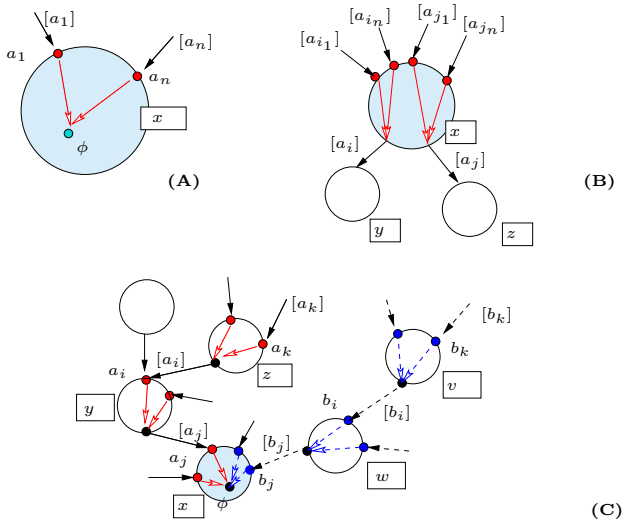

Fig. 2. (A) An *authorization proof* of ϕ for x is a derivation tree whose leaves are evidence formulae *e.g.*, $a_1 \dots a_n$ which are supported by (global) evidences; (B) *accountability* requires authorization for every communication for which there is evidence that x sent it; (C) *weak data accountability* of ϕ for x, requires a global proof which prove the authorisation of ϕ back along all the communication events; There can be more than one path, as illustrated here in solid and dashed arrows. *Strong data accountability* requires that all paths are accountable.

$\models$ for policy formulae. Recall that the truth value of a policy formula depends on the time, the agent doing the reasoning and the observable actions in the system.

Definition 18 (Semantic evaluation of policy $\phi \in \Phi$). *The semantic function* $\models : \mathcal{G} \times \mathcal{I} \times \mathcal{T} \times \Phi \to \{\mathsf{true}, \mathsf{false}\}$*, denoted* $\mathcal{E} \models^t_a \phi$*, is defined as the least function (*$\mathsf{false} < \mathsf{true}$*) satisfying:*

$$\mathcal{E} \models^t_a \phi \text{ when ever } \mathcal{E} \models^t_a b \text{ says } \phi \text{ to } a \text{ for some } b \in \mathcal{G} \tag{9}$$

$$\mathcal{E} \models^t_a \phi[D] \text{ when ever } \mathcal{E} \models^t_a a \text{ owns } d \text{ for all } d \in D \tag{10}$$

$$\mathcal{E} \models^t_a \phi \vee \psi \text{ exactly when } \mathcal{E} \models^t_a \phi \text{ and } I \models^t_a \psi \tag{11}$$

$$\mathcal{E} \models^t_a \phi \wedge \psi \text{ exactly when } \mathcal{E} \models^t_a \phi \text{ or } \mathcal{E} \models^t_a \psi \tag{12}$$

$$\mathcal{E} \models^t_a \phi \to \psi \text{ exactly when } \mathcal{E} \models^t_a \phi \text{ implies } I \models^t_a \psi \tag{13}$$

$$\mathcal{E} \models^t_a a \text{ owns } d \text{ when ever } \mathsf{creates}(t', a, d) \in \mathcal{E} \text{ for some } t' < t \tag{14}$$

$$\mathcal{E} \models^t_a b \text{ says } \phi \text{ to } a \text{ when ever } (\mathsf{comm}(t', b \Rightarrow a, \psi)) \in \mathcal{E} \text{ for some } t' < t \tag{15}$$

One can construct $\models$ basically by building it starting from what follows directly from the evidence set (the last 2 rules) and then repeatedly adding formulae using the other rules. A complication with implication requires that this construction is done by induction on the number

of implications in a formula. We omit further details of this construction. Note that agents "do not care" about communications and data of other agents; For instance, formula b says ϕ to c will not be valid for a other than b or c, *unless* somebody explictly tells a about this (e.g., by c says (b says ϕ to c) to a. However, even in this case a is not able to use ϕ.)

We have that our logic is sound for this semantics.

Theorem 19 (Soundness). *If $\mathcal{P}$ is a authorization proof of a for ϕ at time t, then $\mathcal{E} \models^t_a \phi$.*

A proof is provided in the appendix.

6 Conclusions and Future Work

We have presented a logic for data access and agent accountability in a distributed, heteroeneous setting in which data can be created, distributed and re-distributed. This framework can be used for distributing personal data as well as valuable digital assets. In our system, the owner of the data attaches a usage policy to the data, which contains a logical specification of what actions are allowed with the data, and under which conditions it can be (re-) distributed. This logic allows for different kind of accoutability. We have also demonstrated the soundness of the logic.

We are working on extensions of our system, which can be explained as follows. Suppose Alice gives to BigBrother her personal data d together with a policy ϕ; ϕ might allow BigBrother to re-sell d to BigSister with a policy ϕ'. In our setting ϕ must incorporate ϕ' in some way. In other words, ϕ' must be determined by Alice (the owner of the content) in the first place. In a more realistic scenario, however, BigBrother might legitimately want to supply a ϕ' devised by himself, and what we should check is whether ϕ' complies with Alice's wishes (encoded in ϕ). For instance ϕ might say that each time that BigBrother resells d to someone, Alice should receive a dollar, so everything we should check about ϕ' is whether ϕ' has such a provision. The crucial feature of this extension is that of allowing non-owners to define policies on a content, provided that these policies are in accordance to the owner's wishes. In the current system, policies can only be created by the owner of the data. For the moment, the only way to allow another agent to create their own policy is to make them (co)owner of the data.

To achieve this goal, first we have to extend the logic in two ways: First, we have to incoporate mor complex *conditions* (this is rather straightforward). Conditions will allow policies to refer to groups of agents (e.g., *every "x" satisfying ...* "); for this we basically have to extend our logic

with variables and quantifiers. The use of conditions should allow us to model policies such as the *chinese wall* security policy. The second extension will consist in the incorporation *obligations* (e.g., the the obligation to pay the creator a dollar for each used/resent/... / or to notify the creator/owner if the data is resent/ resold/, etc.). This will be done by extending the notion of observable action. Once conditions and obligations are in place, we can tackle the problem of allowing non-owners to define a policy; in the example above, the crucial condition we need to check is that ϕ' is not more liberal than ϕ; e.g., that each time that ϕ' allows for an action under certain conditions and obligations, then (a derivative of) ϕ allows the same action under the same conditions and obligations.

6.1 Related Work

The literature on access control is so vast that we can only mention the works that are most related to our. One of the earliest proposals is the Access Matrix model (AM), introduced by Lampson [13] and further investigated by Harrison et al. [11]. To provide more flexibility, AM was extended by Samarati and De Capitani di Vimercati [16] by allowing to transfer object privileges but only when the object has an associate copy flag for that privilege. By contrast, in our approach a subject can transfer privileges to other subjects, even if it does not have that right, gaining in flexibility. Abadi presents in [1] a logic based method to represent the AM model. In that work, the subjects can make statements or delegate part of their rights to others. This is somewhat similar to our work, differing in the fact that we introduce a policy formula (a says ϕ to b), whose emphasis is in the 'target agent', to whom that statement is intended. Appel and Felten [2] propose a distributed authentication framework based on proof-carrying proofs from a higher order logic. The agents are authenticated and authorized to access other users' resources, based on the proofs they construct (similarly to a centralized approach). On the other hand, our proposal is *decentralized*, with the data and usage policies flowing between the agents. Moreover, proofs of accountability are only required when a specialized authority inquires a proof, and not continuously. More similar to ours is the work of DeTreville [10], introducing the logic based language Binder, which is specifically designed to express statements in a distributed system. In that work, differently from ours, the statements from any Binder context can be exported and imported to any other context. This implies a total network connectivity, which we do not require. Moreover, as we already mentioned, we explicitly specify the target agent, using the 'to' keyword in the 'says' construction. In the work of Sandhu and Samarati [17], the importance of auditing and the

decentralized administration of authorizations is discussed. Indeed, these issues are also relevant to our approach. In a similar vein, Blaze et al. [8] study trust-management systems. These systems support, like in our approach, delegation and policy specifications. The recent work of Chun and Bavier [9] presents an approach to continuously monitor the trust relations over time, and the use of accountability to check the behaviour of users along a chain of trust. However, implementing this approach is expensive and sometimes infeasible. On the other hand, our lightweight approach can be easily deployed, thanks to the fact that we avoid the monitoring of agents.

Related work for pushing the information in a distributed environment has been proposed by Bertino et al. [7]. In their approach a when a user/agent receives some information, she might send it further to other users/agents without any traceable proof. By contrast, in our work, after an agent receives the data, it might send it to other agents, but along with a usage policy, supporting therefore accountability. Bertino et al. [6] have proposed a logical flexible framework for modelling access control mechanisms. In their work, differently from ours, a logical language is defined for analyzing access control mechanisms and not for describing a new one.

An approach similar to that we proposed here is given by the languages for Digital Right Management. In particular we should mention the eXtensible rights Markup Language (XrML) (`www.xrml.org`) and the Open Digital Rights Language (ODRL) (`www.odrl.net`). For these languages, some formal semantics have been devised, in particular for ODRL by Pucella and Weismann [14, 15]; differently from our framework, these semantics focus on the functionalities of the right languages, and not on the re-distribution mechanism.

As mentioned in the introduction, our system can – thanks to its distributed nature – be used for privacy protection (for protecting private data). A system for specifying and internally enforcing privacy policies and authorizations inside an enterprise (E-P3P) has been proposed by Karjoth et al. [12]. The work was further investigated by Ashley et al. [3], which have also introduced the Enterprise Privacy Authorization Language(EPAL) [4]. Backes et al. [5] continued the research of designing and managing privacy policies in an enterprise, especially for the EPAL language. However, differently from our *decentralized* proposal the E-P3P and EPAL are more suitable for a centralized approach, in which the users are forced to accept the policy of the company.

References

[1] M. Abadi. Logic in access control. *18th IEEE Symposium on Logic in Computer Science*, June 2003.

[2] A. W. Appel and E. W. Felten. Proof-carrying authentication. *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 52–62, November 1999.

[3] P. Ashley, S. Hada, G. Karjoth, and M. Schunter. E-p3p privacy policies and privacy authorization. *Proceeding of the ACM workshop on Privacy in the Electronic Society*, 2002.

[4] P. Ashley, S. Hada, C. Powers, and M. Schunter. Enterprise privacy authorization language(epal). *Research Report 3485, IBM Research*, 2003.

[5] M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. *Computer Security - ESORICS 2003, 8th European Symposium on Research in Computer Security*, 2003.

[6] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Transactions on Information and System Security (TISSEC)*, 2003.

[7] E. Bertino and E. Ferrari. Secure and selective dissemination of xml documents. *ACM Transactions on Information and System Security (TISSEC)*, 2002.

[8] M. Blaze, J. Feigenbaum, and A. D. Keromytis. The role of trust management in distributed systems security. *Secure Internet Programming, Security Issues for Mobile and Distributed Objects*, pages 185–210, 1999.

[9] B. N. Chun and A. C. Bavier. Decentralized trust management and accountability in federated systems. *37th Hawaii International Conference on System Sciences*, January 2004.

[10] J. DeTreville. Binder, a logic-based security language. *IEEE Symposium on Security and Privacy*, pages 105–113, May 2002.

[11] M. H. Harrison, W. L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of ACM*, 19(8):461–471, August 1976.

[12] G. Karjoth, M. Schunter, and M. Waidner. Platform for enterprise privacy practices: Privacy-enabled management of customer data. *Privacy Enhancing Technologies*, 2002.

[13] B. W. Lampson. Protection. *In Proc. Fifth Princeton Symposium on Information Sciences and Systems*, pages 437–443, March 1971. Reprinted in Operating Systems Review, 8,1, January 1974, pp. 18-24.

[14] R. Pucella and V. Weissman. A logic for reasoning about digital rights. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 282–294, 2002.

[15] R. Pucella and V. Weissman. A formal foundation for odrl. In P. Ryan, editor, *Proc. Workshop on Issues in the Theory of Security (WITS)*, 2004.

[16] P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. *Foundations of Security Analysis and Design(Lecture Notes in Computer Science)*, 2171:137–196, 2001.

[17] R. S. Sandhu and P. Samarati. Authentication, access control, and intrusion detection. *The Computer Science and Engineering Handbook*, pages 1929–1948, 1997.

A Soundness Proof

Before showing our soundness result, we need to introduce the notion of *subproof*.

Definition 20. *Let $\mathcal{P}$ be a proof of ϕ for a at time t and let ψ be one of the premises of the last rule used in $\mathcal{P}$. The subtree of $\mathcal{P}$ with root ψ is called a subproof (for ψ) of $\mathcal{P}$.*

This is illustrated in Figure 3. We obtain the following intuitive result:

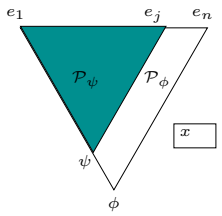

Fig. 3. A proof for ϕ and its subproof of ψ.

Lemma 21. *If $\mathcal{P}$ is an authorization proof for ϕ at time t then any subproof $\mathcal{P}'$ for ψ is an authorization proof for ψ at time t.*

This lemma follows from Definition 20 and the fact that premises of a subproof are also premises of the proof. Now we can obtain our soundness result:

Theorem 22 (Soundness). *If $\mathcal{P}$ is an authorization proof ϕ for a at time t, then $\mathcal{E} \models^t_a \phi$.*

Proof. We proceed by induction on the depth of $\mathcal{P}$. Consider first the base case, where the proof consists of only one step. Take cases on ϕ.

- Case $\phi = a$ owns d. Then the rule must be a `OWN` rule, with premise $\mathsf{creates}(t', x, d)$, with $t' < t$. As $\mathcal{P}$ is an *authorization* proof we have that this premise must be in $\mathcal{E}$. By requirement 14 of Definition 18, we obtain the claim.
- Case $\phi = a$ says ψ to b. Then the rule must be a `COMM` rule, with premise $\mathsf{comm}(t', a \Rightarrow b, \psi[D])$, with $t' < t$. Similar to the previous case, we obtain the claim by requirement 15 of Definition 18.

For the inductive case, we take cases on the last step of $\mathcal{P}$. Consider $\wedge$`I`: We have $\phi = (\psi_1 \wedge \psi_2)$, with premises ψ_1 and ψ_2. By Lemma 21, we have two subproofs $\mathcal{P}_1$ for ψ_1 and $\mathcal{P}_2$ for ψ_2 which are authorization proofs. By induction we have that both $\mathcal{E} \models^t_a \psi_1$ and $\mathcal{E} \models^t_a \psi_2$ thus by requirement 12 of Definition 18 we obtain the claim.

The other cases are similar.