Towards a theory of accountability and audit*

Radha Jagadeesan[1], Alan Jeffrey[2], Corin Pitcher[1], and James Riely[1,**]

[1] School of Computing, DePaul University
[2] Bell Labs, Alcatel–Lucent

Abstract. Accountability mechanisms, which rely on after-the-fact verification, are an attractive means to enforce authorization policies. In this paper, we describe an operational model of accountability-based distributed systems. We describe analyses which support both the design of accountability systems and the validation of auditors for finitary accountability systems. Our study provides formal foundations to explore the tradeoffs underlying the design of accountability systems including: the power of the auditor, the efficiency of the audit protocol, the requirements placed on the agents, and the requirements placed on the communication infrastructure.

1 Introduction

The context of our paper is authorization in distributed systems. The attackers that we consider are untrustworthy principals running arbitrary programs on the network. Attackers may not respect the policies of a system; for example, attackers may create authorization objects without actually having the rights to create them, aiming to subvert the global authorization policy. Traditionally, authorization policies are enforced by controls imposed before shared resources are accessed.

Recently, there has been great interest in accountability mechanisms that rely on after-the-fact verification (Weitzner et al. 2007). In this approach, audit logs record vital systems information and an auditor uses these logs to identify dishonest principals and to assign blame when there has been a violation of security policy. The fear of being "caught" helps to achieve security by deterrence, in the spirit of traditional law enforcement and organizational security. Accountability plays a critical role in the development of trust during human interaction (Friedman and Grudin 1998). Thus, accountability is viewed both as a tool to achieve practical security (Lampson 2004) and as a first-class design goal of services in federated distributed systems (Yumerefendi and Chase 2004).

While designing for accountability is subtle in general (Eriksén 2002), mechanisms to instrument systems to support accountability have been explored in several specific applications: determinate distributed systems (Haeberlen et al. 2007), network storage (Yumerefendi and Chase 2007), validating ISP quality of service claims (Argyraki et al. 2007), internet protocol (Andersen et al. 2008) and policy enforcement on shared documents (Etalle and Winsborough 2007).

In comparison to *a priori* approaches such as access-control, however, the accountability approach to security lacks general foundations for models and programming.

* This is a somewhat fuller version of the paper published in ESORICS2009.

** Supported by NSF Career 0347542.

Citing a small sample of references, access control has (a) *operational* models in the form of automata (Schneider 2000), with associated algebraic models based on regular expressions (Abadi et al. 2005); (b) *logic-based declarative* approaches in a fragment of many-sorted first-order predicate logic (Halpern and Weissman 2003; Li and Mitchell 2003); and (c) *static analysis* to validate the access-control properties of interfaces, e.g., types for authorization (Fournet et al. 2005; Cirillo et al. 2008).

In this paper we make two contributions toward bringing such formal foundations to the study of accountability. First, we describe an operational model of accountability based systems. Honest and dishonest principals are described as agents in a distributed system where the communication model guarantees point-to-point integrity and authenticity. Auditors and other trusted agents (such as trusted third parties) are also modeled internally as agents. Behaviors of all agents are described as processes in a process algebra with discrete time. Auditor implementability is ensured by forcing auditor behavior to be completely determined by the messages that it receives.

Second, we describe analyses to support the design of accountability systems and the validation of auditors for finitary systems (those with finitely many principals running finite state processes with finitely many message kinds). We compile finitary systems to (turn-based) games and use alternating temporal logic to specify the properties of interest. This permits us to adapt existing model-checking algorithms for verification.

Our results provide the foundations necessary to explore tradeoffs in the design of mechanisms that ensure accountability. The potentially conflicting design parameters include the efficiency of the audit, the amount of logging, and the required use of message signing, watermarking, or trusted third parties. Design choices place constraints on the auditor, the agents of the system and the underlying communication infrastructure.

The paper is organized as follows. We motivate our approach in Section 2. Section 3 describes the model and Section 4 describes the analysis framework. The ideas are illustrated using examples in Section 5. We survey related work in Section 6. In this extended abstract, we elide all proofs.

2 Overview of our approach

In this section, we illustrate the motivations behind the design of our framework using variants of a motivating example from (Barth et al. 2007).

In Section 5, we analyze an abstract variant of the example that permits message forwarding amongst health professionals. Our analysis yields a variety of auditors for the example, even in general distributed settings, and shows that powerful mechanisms, such as trusted third parties, are not necessary for all audit protocols.

Example 1 (My Health). The MyHealth patient portal at Vanderbilt University Hospital allows patients to interact with healthcare professionals through a web based system. There are three possible roles that can be assumed by principals: health professionals (doctors and nurses), non-health professionals (secretaries), and patients. The possible messages include health questions from patients and health answers from doctors. We focus on the two privacy policies in Barth et al. (2007): (a) a health question can only be directed to a health professional, and (b) a health answer about a patient can only

be directed to the same patient or to a health professional. These policies permit health professionals to forward health information amongst themselves. In the discussion below, we will consider the case where patient Charlie contacts the auditor because he has received a health answer from doctor Bob that was intended for a different patient. The motivation for such an audit is to aid in the detection and discovery of the source of the leak. □

We now describe our model and its relation to the following properties. The discussion is intended to establish intuitions, with formalities defered to later sections.

- *Upper bound:* Every agent guilty of a dishonest action is blamed by the auditor.
- *Lower bound:* Everyone blamed by the auditor is guilty.
- *Overlap:* At least one of the agents blamed by the auditor is guilty.
- *Liveness:* The auditor is always successful in blaming a non-empty subset of agents.
- *Blamelessness:* Honest agents have a strategy to avoid being pronounced a *possible* offender by an auditor.

Agents. We model the behavior of principals (both honest and dishonest) as agents in a distributed system. Auditors are also modeled internally as honest agents. We use processes to specify an upper bound on honest behavior: a principal is behaving honestly in a run whenever their contribution to the run is a trace of an honest process. A dishonest agent is unconstrained. A run of an agent reveals its dishonesty if it is not a permitted trace for an honest agent.

The communication model captures point-to-point communication over an underlying secure communication mechanism which provides integrity and authenticity guarantees, but provides no additional mechanisms for non-repudiation or end-to-end security. This model is realizable using transport mechanisms such as TLS.

Dishonest agents may collaborate arbitrarily. This means that the auditor has to achieve its objectives independent of potential cartels of dishonest agents. Honest agents may also collaborate, depending upon the specification of honest agents.

Internal auditors. Auditors are intended to be realizable agents in a distributed system without global knowledge. Thus, they are unaware of transactions that do not involve them, and their local state is only influenced by the messages that they receive. In contrast, the strategies adopted by dishonest agents can potentially depend on viewing traffic on the network between other agents. The internalization of auditors limits them. Auditors can only address dishonest behaviors using the information available on individual runs of a system; they cannot audit violations of security properties that need sets of traces for their specification (such as non-interference). Auditors cannot detect cartels of dishonest agents who conduct dishonest exchanges amongst themselves.

Thus, our auditors cannot in general satisfy *Upper bound*. To see this, consider the leakage of patient records to a dishonest non-health professional by a dishonest health professional via out-of-band mechanisms without using the MyHealth website in Example 1. Such leakage of records by dishonest agents solely to dishonest agents will not be detected at all by an auditor in our framework.

Mandatory logging and responsiveness. Even in the case that the auditor has become aware of dishonest behavior and initiates an audit, the auditor is powerless unless there are statutory and enforceable reporting requirements on the honest agents.

In Example 1, if there is no requirement for maintaining and presenting records, doctor Bob can achieve "absence of provable guilt" by maintaining no records. Such reasoning motivates requirements on honest agents to maintain audit logs in several accountability systems. Furthermore, a guarantee that honest principals provide answers to audit queries is needed for the auditor to achieve *Liveness*.

These desiderata motivate the inclusion of time in our system specification formalism to enable systems to mandate promptness on honest agents. Thus, auditors can use tardiness as evidence for dishonesty and assign blame to such tardy agents. Our model uses *discrete time*, which is abstract and logical rather than quantitative.

Communication model. The absence of non-repudiation in our basic communication model limits the accuracy of the audit process to *Overlap*. For example, in the audit scenario of Example 1, the auditor commences by querying doctor Bob: if Bob disagrees that he sent the message to patient Charlie, the auditor can deduce that at least one of Bob or Charlie is compromised. The absence of non-repudiation prevents further disambiguation. Alternately, Bob might point to another principal, Eve, as the sender of the patient health message. In this case, the auditor proceeds to question Eve. This process either ends in one of two ways. (a) The auditor discovers two principals (perhaps one of whom is a health professional) who disagree on messages sent by one and received by the other, as sketched above; in this case, both the principals are deemed guilty. (b) The auditor discovers a cycle of non-health professionals, each claiming to have received the message from the predecessor in the cycle; in this case, the entire cycle of principals is deemed guilty. In either case, the auditor achieves *Overlap*.

This situation may be unsatisfactory to an honest agent, since it is not possible for an honest agent to achieve *Blamelessness*. In addition the auditor cannot achieve stronger properties for the auditor, such as *Lower bound*. Such properties require more detailed and secure logging of messages.

We do not limit attention to strong models of communication—such as those enabling non-repudiation—because weaker models are often more realistic. For example, in the IETF Session Initiation Protocol (SIP), a typical SIP proxy is expected to handle large volume of calls; thus, it is difficult to successfully mandate computationally expensive signature based methods on each point-to-point communication link.

As evidence for the flexibility of our modeling, we show that our model can indeed encode notaries as trusted third parties. This permits us to address the stronger communication guarantees required to accurately capture examples such as the MyHealth website of Example 1.

3 Formalizing the model

Based on a notion of process, defined below, we will define an arena $\langle \mathscr{A}, \mathscr{M}, \mathscr{H} \rangle$ to be a set $\mathscr{A}$ of principals, a set $\mathscr{M}$ of messages, and a set $\mathscr{H}$ of processes, which define the honest behaviors of agents. Later, we shall give example arenas, and then give desirable properties of auditors in such an arena.

Our formal model is based on Communicating Sequential Processes (Brookes et al. 1984), I/O automata (Lynch 2003), and discrete timed process algebra (Hennessy and Regan 1995). Our processes are *input-enabled*, to prevent a (perhaps dishonest) agent

from blocking the output of other agents. We use *discrete time*, and the *timeouts* that it engenders, to specify conditions on prompt response. Our communication model provides integrity and authenticity guarantees but provides no additional mechanisms for non-repudiation or end-to-end security. We use processes as a *safety* specification of honest behavior: a principal is behaving honestly in a run whenever their contribution to the run is a trace of an honest process.

Actions. Fix a countable set $\mathscr{A}$ of *principals* and a countable set $\mathscr{M}$ of *messages*. Let a, b, c, d, h range over elements of $\mathscr{A}$; A, B, C, D, H over subsets of $\mathscr{A}$; and m over elements of $\mathscr{M}$.

The set of *actions* $\mathscr{K}$ over $\langle \mathscr{A}, \mathscr{M} \rangle$ is then generated by the grammar

$$k, \ell ::= a{\to}b{:}m \mid \sigma$$

where $a{\to}b{:}m$ represents a message m sent from a to b, and σ represents a timeout.

Relative to a set of principals A, an action may be output, input, internal, disjoint or timeout. The action $(a{\to}b{:}m)$ is *output* from A if $a \in A$ and $b \notin A$, *input* to A if $a \notin A$ and $b \in A$, *internal* to A if $a \in A$ and $b \in A$, and *disjoint* from A if $a \notin A$ and $b \notin A$. The action σ is timeout from A, for any A.

We often describe actions from the point of view of a particular principal, using ? for inputs and ! for outputs. Thus, when giving the example of a process for a, we will write $a{\to}b{:}m$ as $a{\to}b!m$ and $b{\to}a{:}m$ as $b{\to}a?m$.

Processes. A *process* over $\langle \mathscr{A}, \mathscr{M} \rangle$ is a quadruple $P = \langle A, S, s_0, \longrightarrow \rangle$ where (a) $A \subseteq \mathscr{A}$ is a subset of principals, (b) S is a set of states, ranged over by s and t, (c) $s_0 \in S$ is a distinguished start state, (d) $\longrightarrow \subseteq S \times \mathscr{K} \times S$ is a labeled transition relation in which labels are actions over $\langle \mathscr{A}, \mathscr{M} \rangle$. We call A the *principals of* P, written $\pi(P)$.

We say that s allows k whenever there exists a t such that $s \xrightarrow{k} t$. We also require that no label in $\longrightarrow$ is disjoint from A, every state in S allows every input for A (input-enabling), every state in S allows at least one timeout or output for A (timeout-enabling).

Whenever A and B are disjoint we define the *composition* of processes $P = \langle A, S, s_0, \longrightarrow_1 \rangle$ and $Q = \langle B, T, t_0, \longrightarrow_2 \rangle$ to be $P \parallel Q = \langle A \cup B, S \parallel T, s_0 \parallel t_0, \longrightarrow \rangle$ where $S \parallel T = \{(s \parallel t) \mid s \in S \text{ and } t \in T\}$ and $\longrightarrow$ is defined as follows:

$$\frac{s \xrightarrow{k}_1 s'}{s \parallel t \xrightarrow{k} s' \parallel t} \; k \text{ is disjoint from } B \qquad \frac{t \xrightarrow{k}_2 t'}{s \parallel t \xrightarrow{k} s \parallel t'} \; k \text{ is disjoint from } A$$

$$\frac{s \xrightarrow{k}_1 s' \quad t \xrightarrow{k}_2 t'}{s \parallel t \xrightarrow{k} s' \parallel t'} \; \begin{array}{l} k \text{ is input to } A \text{ and output from } B, \text{ or} \\ k \text{ is input to } B \text{ and output from } A, \text{ or} \\ k \text{ is } \sigma \end{array}$$

We write $\prod_{i \in I} P_i$ for the composition of processes P_i.

A *trace*, v, w, is a finite sequence of actions. Write ε for the empty trace, and $v.w$ for trace composition. Write $s \stackrel{v}{\Longrightarrow} s'$ to indicate that there exists a sequence of transitions from s to s' labeled by v. A trace has principals A whenever it contains no action disjoint from A. A *run* of a process P with start state s_0 is a trace v such $s_0 \stackrel{v}{\Longrightarrow} s'$ for some s'. Note that any run of a process with principals A must have principals A.

Write $v{\restriction}A$ for the projection of v onto actions relating to A:

$$\varepsilon{\restriction}A = \varepsilon \qquad v.(b{\to}c{:}m){\restriction}A = v{\restriction}A \qquad \text{if } A \cap \{b, c\} = \emptyset$$

$$v.\sigma\restriction A = v\restriction A.\sigma \qquad v.(b{\to}c{:}m)\restriction A = v\restriction A.(b{\to}c{:}m) \text{ if } A\cap\{b,c\}\neq\emptyset$$

Note that for any P with principals A and Q with principals B, v is a run of $P \parallel Q$ whenever v has principals $A \cup B$, $v\restriction A$ is a run of P, and $v\restriction B$ is a run of Q. This is the usual trace semantics of parallel composition in CSP (Brookes et al. 1984).

Arenas. An *arena* $\langle \mathscr{A}, \mathscr{M}, \mathscr{H} \rangle$ comprises a countable set $\mathscr{A}$ of principals, a countable set $\mathscr{M}$ of messages, and a countable set $\mathscr{H}$ of process over $\langle \mathscr{A}, \mathscr{M} \rangle$.

Given a principal set H and a process P, we say that H is *honest* in P if $P = \prod_{i\in I} P_i$ and for all $h \in H$ there exists $P_i \in \mathscr{H}$ such that $\pi(P_i) = \{h\}$, that is, if every honest principal must be represented by an honest process.

We note that honesty for processes is down-closed (if $H \supseteq H'$ and H is honest in P then H' is honest in P) and union-closed (if H and H' are honest in P then so is $H \cup H'$); so any process has a maximum honest set of principals.

Given a principal set H and a trace v, we say that H is *honest* in v whenever there exists a process P with run v such that H is honest in P. Honesty for traces is down-closed and union-closed; so any trace has a maximum honest set of principals.

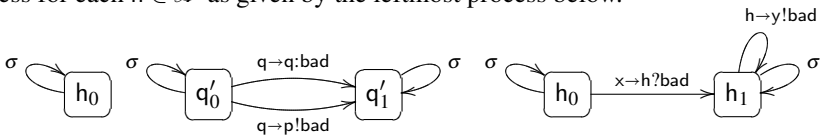
Honesty is a global property of traces, that is for any trace v capturing all behavior of the system, we can determine the principals H who have behaved honestly in that trace. The problem of audit is that the auditor is not provided with the trace v but only a local fragment of v.

Example Arenas. In the following examples, we write principal names without subscripts or superscripts (p, h, b) and write states belonging to a principal with numeric subscripts and optional primes (p_0, h''_5, b'_2) with the convention that p_i is a state of an honest process with principals $\{\mathsf{p}\}$, and that p_0 is the start state of the process.

Honesty and dishonesty are properties of principals with respect to a trace, rather than an intrinsic property of a principal. Nonetheless, we find it helpful to use suggestive names in examples to indicate principals that are intended to be honest or dishonest. We use p, q, r for general principals; h, g, f for principals with honest behaviors; and d, c, b for principals with dishonest behaviors. We also use x, y, z for parameters and a for auditors, as discussed below.

We elide transitions required solely for input enabling, assuming an implicit transition $\mathsf{p}_i \xrightarrow{k} \mathsf{p}_i$ for any input action of p that is not explicitly given.

Example 2. Consider an arena with $\mathscr{A} = \{\mathsf{p},\mathsf{q}\}$ and $\mathscr{M} = \{\mathsf{bad}\}$, and define an honest process for each $\mathsf{h} \in \mathscr{A}$ as given by the leftmost process below.



The initial state $\mathsf{p}_0 \parallel \mathsf{q}_0$ shows that both p and q are honest in any trace containing only timeouts.

p is also honest in the traces q→q:bad (because the action is disjoint with p) and q→p:bad (because p is input enabled), as witnessed by the initial state $\mathsf{p}_0 \parallel \mathsf{q}'_0$ where q'_0 is defined by the center process above. q is dishonest in these traces, since there is no process that is honest for q which allows them. Moreover, q is dishonest in any trace

containing q→q:bad or q→p:bad. Symmetrically, q is honest in p→p:bad and p→q:bad, whereas p is dishonest in these traces.

Auditing in this arena is trivial, since the sender of a bad message is guaranteed to be dishonest. This example corresponds to the case in Example 1 when no principal is allowed to forward health answers to anyone except the patient in question.

There is a problem with initiating audit, however, in that honest agents have no mechanism for reporting dishonest behavior, for example p cannot report the receipt of the message q→p:bad to an auditor. □

Next, we model the message forwarding capabilities of principals in Example 1.

Example 3. In the variant given by the rightmost process above, honest processes are allowed to forward the first bad message that they receive; for example, reporting dishonest behavior to an auditor. If an auditor knows that a message p→q:bad has been sent, then there must be a dishonest principal, but does not know who is dishonest – there are traces containing p→q:bad in which p is honest, or q is honest, or both. The goal of the auditor should be to determine the agent that *initiated* the bad message. □

Auditors. An *arena with audit* is an arena with a distinguished honest principal a and a set of distinguished messages blame B for every $B \subseteq \mathscr{A}$, indicating the blame set B. For simplicity, we treat the blame action as internal to the auditor, and thus we abbreviate the action "a→a:blame B" as "a:blame B".

We now consider various notions of correctness for auditors. Many of these notions, while appealing, have serious technical problems, and so we will not consider them further in this paper.

In these definitions, we will discuss a trace with dishonest principals D, defined to be $\mathscr{A} \setminus H$ where H is the largest honest set.

Candidate 1 (Upper bound). *An arena with audit provides an* upper bound on dishonesty *if, for any trace v with dishonest $D \not\ni$ a containing* a:blame B*, we have $D \subseteq B$.*

Unfortunately, the only auditor capable of providing an upper bound on dishonesty is one which blames all principals who are capable of dishonesty, regardless of whether they acted dishonestly or not.

$A \subseteq \mathscr{A}$ are said to be *capable of dishonesty* in an arena whenever there is a trace v internal to A (i.e., all messages from $a \in A$ are sent to some $b \in A$) with dishonest A.

Proposition 4. *In any arena where audit provides an upper bound on dishonesty, and where $A \not\ni$ a are capable of dishonesty, we have that any trace containing* a:blame B *must have $A \subseteq B$.*

We do not consider this notion of correctness further.

Candidate 2 (Lower bound). *An arena with audit provides an* lower bound on dishonesty *if, for any trace v with dishonest $D \not\ni$ a containing* a:blame B*, we have $B \subseteq D$.*

Unfortunately, auditors are only capable of blaming dishonest principals who confessed their own dishonesty. Dishonest principals who do not confess will never be blamed.

In a trace v with dishonest $D \ni d$, we say that d *confessed* whenever, for any w such that $v{\restriction}\{d,\mathsf{a}\} = w{\restriction}\{d,\mathsf{a}\}$ we have that w has dishonest $D' \ni d$.

Proposition 5. *In any arena where audit provides a lower bound on dishonesty, and any trace containing* a:blame B *with* $d \in B$*, we have that d confessed.*

Trust mechanisms (such as trusted third parties) are required to establish the non-repudiation implied in the above proposition. We discuss these in Section 5.

Candidate 3 (Overlap). *An arena with audit provides* overlap with dishonesty *if, for any trace v with dishonest* $D \not\ni \mathsf{a}$ *containing* a:blame B*, we have* $B \cap D = \emptyset$ *implies* $B = \emptyset$*.*

Overlap is a more general property than providing a finite lower bound, since any lower bound $\{d_1, \ldots, d_n\}$ can be replaced by a series of singleton overlaps $\{d_1\}, \ldots, \{d_n\}$.

We do, however, note one problem with this definition, which is that although it is up-closed, it is not intersection-closed, that is there may be v.a:blame $B.w$ and v.a:blame $C.w$ which overlap with dishonesty, but v.a:blame $(B \cap C).w$ does not. This may arise in cases of separation of duty (Ferraiolo et al. 2003), if p and q must dishonestly collude to cause some action, then an auditor might choose to blame either $\{\mathsf{p}\}$ or $\{\mathsf{q}\}$, but not $\emptyset$. We leave this problem for future work.

Candidate 4 (Liveness). *An arena with audit is n-live if for any run* $v.k.w$ *such that* a *is honest, k is an input to* a*, and w contains at least n timeout actions, there is an action* a:blame B *in w. An arena with audit is live whenever it is n-live for some n.*

As is common with correctness criteria, we distinguish between safety properties and liveness properties. In this case, liveness is quite simple to specify and verify (since an arena is n-live precisely when the honest processes for a are n-live).

4 Analysis using turn-based games

This section describes the use of game-based methods to automate the analysis of the properties described in the prior section. We refer the reader to (Alur et al. 2002) for background motivation and detailed examples.

Definition 6. A *turn-based game graph* over n-players player 1 to player n is $\mathscr{G} = (q, \mathscr{S} = \mathscr{S}_1 \uplus \cdots \uplus \mathscr{S}_n, \mathscr{E}, \Pi, \pi)$ where:

- $(\mathscr{S}, \mathscr{E})$ is a directed graph with a total transition relation $\mathscr{E}$ over the finite stateset $\mathscr{S}$.
- $\mathscr{S}_1, \ldots, \mathscr{S}_n$ is a partition of $\mathscr{S}$ and $q \in \mathscr{S}$ is the start state
- Π is a set of propositions; $\pi : \mathscr{S} \to \Pi$ yields the propositions true at each state. □

An evolution proceeds as follows. States in $\mathscr{S}_i$ are *player-i* states, where player i decides the successor state. A path in the game graph is a finite or infinite sequence of states. By totality, every finite path extends to a *play*, an infinite path of states.

Strategies. A (pure) strategy for a player is a recipe to extend a play, i.e., given a finite sequence of states, representing the history of the play, a strategy for a player chooses a unique successor state to extend the play.

Let mem_i be a set called *memory* that encodes the information about the history of the play. A player i strategy can be described as a pair of functions: a *memory-update*

function ς^{U}: $2^{\Pi_i} \times mem_i \to mem_i$ to update the memory with the current state and a *next-move* function ς^{M} that yields a new player i move for every element of $\mathscr{S}_i \times mem_i$. A strategy must prescribe only available moves, i.e., for all $s \in \mathscr{S}_i$, for all $m \in mem$, we have $(s, \varsigma^{\mathrm{M}}(s,m)) \in \mathscr{E}$.

Let Σ_i stand for the set of valid player i strategies under consideration. Strategies interact as follows. Player i follows the strategy ς_i if in each player i move, she chooses the next state according to ς_i^{M}. Once a starting state $s \in \mathscr{S}$ and strategies $\varsigma_i \in \Sigma_i$ of the players are fixed, it is clear that the resulting outcome is a play of the game.

Compilation. We compile a finite collection of finite state processes (with a finite universe of messages) into a turn-based game. The translation uses new propositions $guilt_{\mathsf{p}}$ for every $\mathsf{p} \in \mathscr{A}$. If $guilt_{\mathsf{p}}$ is satisfied by a state on a path, then p is dishonest on that path.

Logic. We use a fragment of the logic ATL* (Alur et al. 2002). The usable propositions are restricted to the ones of interest. The path formulas exclude the next modality(as found, for example in LTL-X (Clarke et al. 1999).

As a result, the properties are insensitive to the extra transitions introduced by the above compilation of arenas into turn-based games.

We refer to (Alur et al. 2002) for precise semantics. The state (ϕ) and path (ψ) are given by the following grammar: (A is any subset of principals)

$$\phi ::= \mathtt{t} \mid guilt_{\mathsf{p}} \mid \sigma \mid \mathsf{a}{:}\mathsf{blame}\ B \mid \mathsf{p}{\to}\mathsf{q}{:}m \mid \neg\phi \mid \phi \vee \phi \mid \langle\!\langle A \rangle\!\rangle \psi$$
$$\psi ::= \mathtt{t} \mid \phi \mid \neg\psi \mid \psi \vee \psi \mid \Box\psi \mid \Diamond\psi \mid \psi \mathscr{U} \psi$$

The formula $\langle\!\langle A \rangle\!\rangle \psi$ is true at a state if there exist strategies for the players in set A such that no matter what strategies the other players (in the complement of A) choose, the resulting play satisfies the path formula ψ.

We use existential and universal quantification over finite sets instead of finite disjunction and conjunction; e.g., $(\exists \mathsf{p})\psi$ is shorthand for $(\vee_{\mathsf{p}})\psi$. Also, we define

- $NonZeno \triangleq \Box\Diamond\sigma$, to identify live traces with infinitely many σ actions.
- $AInit \triangleq \exists \mathsf{p}, m.\Diamond(\mathsf{p}{\to}\mathsf{a}{:}m)$, to identify traces where the auditor a has been initialized by being sent some message.
- $Succ(B) \triangleq NonZeno \wedge Ainit \wedge \Diamond(\mathsf{a}{:}\mathsf{blame}\ B)$, to identify Non-Zeno traces where the auditor has been contacted and the auditor has assigned blame to B.

and

Overlap	$\langle\!\langle \emptyset \rangle\!\rangle Succ(B) \Rightarrow (\exists \mathsf{p} \in B) guilt_{\mathsf{p}}$
Lower bound	$\langle\!\langle \emptyset \rangle\!\rangle Succ(B) \Rightarrow (\forall \mathsf{p} \in B) guilt_{\mathsf{p}}$

Since the auditor is fixed, these are LTL properties. Thus, when expressed in ATL*, they have the strategy quantifier with the empty set to capture universal quantification over all traces reflecting other player choices. The soundness of the logical encoding above w.r.t. the trace based definitions of the earlier section follows from the soundness of the compilation w.r.t. the trace semantics of an arenaProposition 11 in Section A).

Blamelessness of p for a fixed audit protocol is true at a state only if the agent a has a strategy to ensure that p never ends up in the blame set assigned by the auditor, independent of the given fixed auditor strategy and independent of any choice of strategies

for the scheduler and the other players. Formally, we define

Blamelessness	$\langle\!\langle \mathsf{p} \rangle\!\rangle \neg(\exists B \ni \mathsf{p}) \Diamond(\mathsf{a{:}blame}\, B)$

The model-checking problem for ATL*is 2EXPTIME in the size of the formula and PTIME-hard for bounded-size formulas (Alur et al. 2002). So, we have:

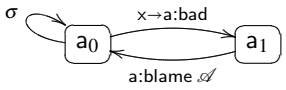
Proposition 7. *The model-checking problem for* Overlap*,* Lower bound *and* Blamelessness *for an arena* $\langle \mathscr{A}, \mathscr{M}, \mathscr{H} \rangle$ *with a fixed audit protocol is solvable in EXPTIME in the size of the arena and 2EXPTIME in the formula size.*

The formulas of interest are small. The bottleneck is the EXPTIME dependence on arenas caused by the determinization of the honest processes in the compilation process.

5 Example auditors

We present a series of examples in which the auditor aims to detect the origin of a special bad message. At the end of this section, we relate the discussion to Example 1.

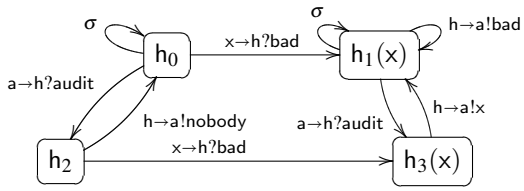
Example 8. Consider auditing the arena in Example 3. When the auditor receives a bad message, they know that there is a dishonest principal. However, since the arena does not permit them to query principals for further information, they have no way to discover the guilty parties. So the best they can do is blame everyone:

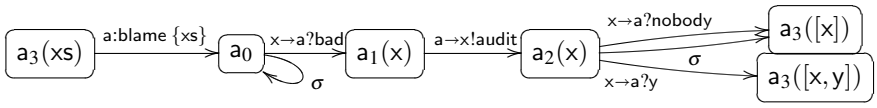


This auditor provides liveness and overlap with dishonesty, albeit trivially. This auditor does not provide lower bound. □

We now consider audit protocols where honest principals are required to respond to requests for information from the auditor. From a principal h, the auditor will request the identity of the sender of a bad message to h. We analyze the variations that arise depending on whether: (a) honest principals may forward bad messages to principals other than the auditor; (b) honest principals are *required* to report bad messages (all are *allowed* to report bad messages); (c) senders report to whom they have forwarded bad.

Example 9. Extend the arena of Example 2 to accommodate audit by setting $\mathscr{M} = \mathscr{A} \cup \{\mathsf{bad}, \mathsf{audit}, \mathsf{nobody}\}$. The honest processes are described below, where we describe the potentially infinite state transition systems using state variables x, y, xs ranging over $\mathscr{A}$. States h_1, a_1 and a_2 are parameterized by principal x, which sent the bad message. State a_3 is parameterized by the principal list xs, which are blamed; we use ML notation for lists ([] for the empty list, :: for prefixing, @ for concatenation).





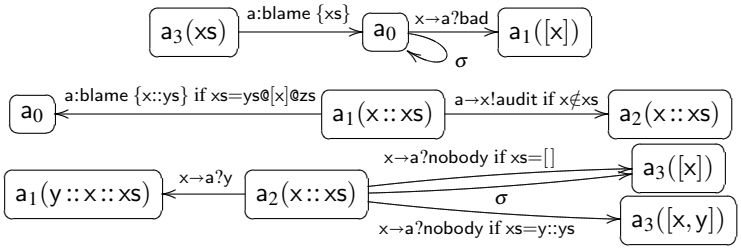
Honest principals may initiate an audit by reporting the receipt of a bad message to the auditor (the transition h→a!bad at h_1). The auditor responds to audit with a request for the sender of the bad message. If the auditor's request (at a_2) times out, or the response is nobody, then the principal initiating the audit is dishonest and is blamed. If the response x→a?y is received, then the auditor blames $\{x, y\}$ because it is unable to detect whether y initiated or forwarded bad, or whether x is lying about receipt of bad from y. This auditor provides liveness and overlap with dishonesty. The algorithms of Section 4 verify this for the case when the number of principals is finite. □

We now analyze the consequences of allowing honest principals to forward bad messages to principals other than the auditor.

Example 10. Allowing honest principals to forward bad messages to all principals, as in Example 3, necessitates changing the auditor to track down the original source of the bad message. To see this, first modify the arena from Example 9 by replacing the h→a!bad self-loop on $h_1(x)$ with h→y!bad.

The auditor from Example 9 does not provide an overlap with dishonesty for this new arena, because a trace of the form d→h!bad, h→g!bad, g→a!bad, ... would result in g and perhaps h being blamed incorrectly (their forwarding behavior is honest) when only d has been dishonest (initially sending bad).

To identify an originator of a bad message (there may be several), the auditor below follows a chain of forwarders until it receives: (a) no response (a timeout); (b) the answer nobody; (c) the answer a; or (d) it finds a cycle of forwarders. The auditor then blames: (a) the principal that did not respond to an audit request (it is dishonest to ignore the auditor); (b) the principal p that responded with nobody and, if there is one, the principal q that claimed p forwarded bad to q (either q is lying about receiving a forwarded bad or p is unable to identify a principal that forwarded bad to them); (c) the principal that claimed a forwarded bad (that principal is lying because the auditor does not send bad); (d) all principals in the cycle (one of them is lying about the source). States a_1, a_2 and a_3 are parameterized by the list of suspected principals.



This auditor provides liveness and overlap with dishonesty. The algorithms of Section 4 verify this for the case when the number of principals is finite. □

It is important in the above example that the arena requires honest principals to record the *initial* sender of bad rather than the most recent sender to make the auditor overlap

with dishonesty. (If instead the *most recent* sender of bad was reported to the auditor, and we saw a trace ending with a cycle of the form d→h!bad, h→g!bad, g→h!bad, h→a!bad, ..., then the auditor would find and blame the cycle h to g to h. Neither g nor h are dishonest, so the auditor above would not overlap with dishonesty for the modified arena.)

In both Example 9 and Example 10, an honest agent is unable to achieve *Blamelessness*. We address this issue next by encoding the use of notaries as trusted third parties to permit honest agents to establish blamelessness.

Notaries. The presence of notaries provides a non-repudiation function and disables the ability of a dishonest principal d to get an honest principal h blamed (by simply claiming that h sent bad to d). The notary principals are assumed to be honest. For this reason, we refer to the notary principals as Trusted Third Parties (TTPs). Here we consider a single non-auditor principal for the sake of simplicity, but it is not essential that there be only one TTP.

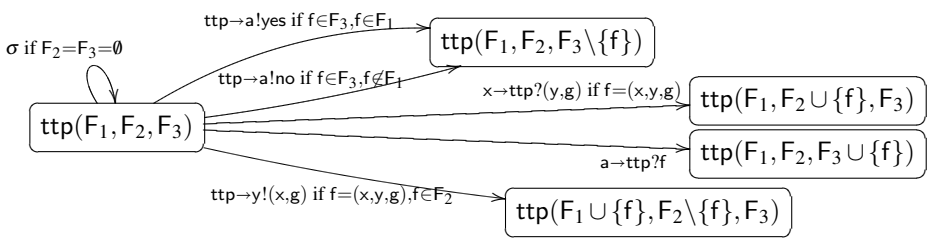
We assume a collection of messages $\mathscr{G} \ni \mathsf{g}$ that pass uninterpreted through the TTP. We define $\mathscr{G}$ to include the bad message with different provenance chains indicating the path of the bad message. With $\mathscr{G}$ fixed, we then define the messages of the arena by:

$$\begin{array}{ll} \mathscr{M} \triangleq (\mathscr{A} \times \mathscr{G}) & \text{(Messages to and from TTP)} \\ \quad \cup (\mathscr{A} \times \mathscr{A} \times \mathscr{G}) & \text{(Message query by auditor to TTP)} \\ \quad \cup \{\mathsf{yes}, \mathsf{no}, \mathsf{unknown}\} & \text{(Response to auditor)} \end{array}$$

We use f to range over *forwarding records* of the form $(\mathsf{x}, \mathsf{y}, \mathsf{g})$ indicating that the TTP forwarded g from x to y. We use F to range over sets of forwarding records.

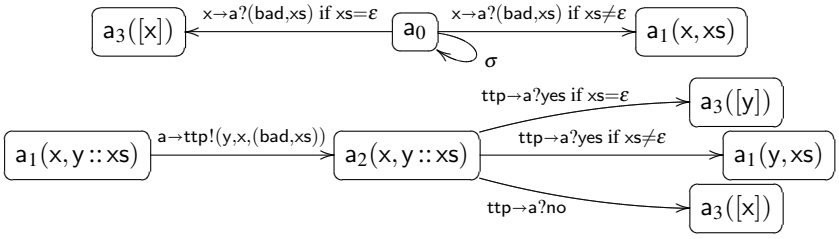
The TTP interacts with principals by forwarding messages on their behalf. A principal x sends a forwarding request of the form (y, g) to the TTP (indicating the target). Subsequently, the TTP forwards the message (x, g) (indicating the source) to y, and adds the forwarding record $(\mathsf{x}, \mathsf{y}, \mathsf{g})$ to its store. The TTP also respond to queries from the auditor that ask whether $\mathsf{f} = (\mathsf{x}, \mathsf{y}, \mathsf{g})$ has been forwarded in the past. It can only respond honestly with yes (resp. no) if its store contains the forwarding record f (resp. does not contain the forwarding record f).

The TTP state $\mathsf{ttp}(\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3)$ is parameterized by three sets of forwarding records. The set F_1 stores which messages have been forwarded. The set F_2 maintains the forwarding requests received but not yet acted upon. The set F_3 maintains the auditor requests received but not yet acted upon. The TTP may only timeout when there are no actions to complete, i.e., $\mathsf{F}_2 = \mathsf{F}_3 = \emptyset$. The sets of actions not yet completed are present to ensure that the TTP is input enabled. The behavior of the TTP is specified as:



Provenance. We add *provenance* information to the messages. In this context, provenance is a sequence (possibly empty) of principal names indicating the path of a forwarded message. An empty provenance sequence indicates that the message was not forwarded, i.e., in $\mathsf{x}{\rightarrow}\mathsf{y}!(\mathsf{bad},\varepsilon)$, x is confessing to sending bad directly. In contrast, a non-empty provenance sequence of the form $(\mathsf{z}::\mathsf{xs})$ indicates that the message was forwarded with z being the most recent forwarder, i.e., in $\mathsf{x}{\rightarrow}\mathsf{y}!(\mathsf{bad},(\mathsf{z}::\mathsf{xs}))$, x is claiming that they received the forwarded message from z as $\mathsf{z}{\rightarrow}\mathsf{x}?(\mathsf{bad},\mathsf{xs})$. We now demand that honest communication between principals occurs via the TTP (operating without knowledge of the provenance structure), and so we define $\mathscr{G} \triangleq \{\mathsf{bad}\} \times \mathscr{A}^*$.

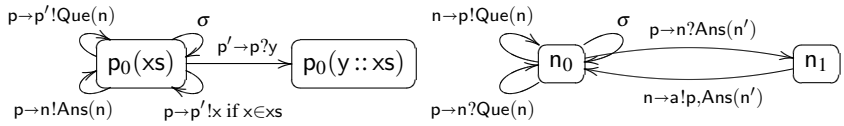
Auditor. When a principal x initiates an audit by sending bad paired with a provenance sequence to the auditor, the auditor can verify the entire provenance sequence step-by-step, using the TTP to determine whether each forward indicated in the provenance sequence is genuine. If the empty provenance sequence is ultimately found, the initial sender is blamed. If the TTP responds with no at any point, then a principal has claimed that it forwarded a message but is unable to prove its claim, that principal is blamed. The auditor is formalized as:



Honest Agents. Honest agent h are constrained as follows. (a) h is required to report bad to the auditor, and (b) h can only forward bad messages that are received via the TTP, after honestly updating the provenance and using the TTP. We elide the straightforward formalization.

Since the auditor is able to verify evidence: (a) Honest agents have *Blamelessness*, and (b) the auditor has *Lower bound*. The algorithms from Section 4 verify these statements when there are finitely many principals and messages and the length of the provenance chain is bounded.

Example 1 revisited. We discuss briefly the implications for Example 1. Let p, p' range over health professionals (doctors and nurses) and n, n' over patients and non-health professionals (secretaries). Let $\mathsf{Que}(\mathsf{n})$ and $\mathsf{Ans}(\mathsf{n})$ be messages representing question and answers concerning patient n. In the following processes, xs represents messages that have been sent to p that may be forwarded.



Honest health care professionals have unrestricted exchange of messages amongst themselves. However their answers and questions to patients are constrained to be about the

receiver. A patient who has received an answer about another patient is allowed to initiate an audit via a message to the auditor.

The model closest to the original example Example 1 is the one from Example 5 since the MyHealth website is effectively a TTP. The techniques of Example 5 permit an auditor permit an auditor to achieve *Lower bound*, and the honest agents to have *Blamelessness*.

Perhaps of greater interest, our analysis in Section 5 shows that even without TTPs, auditors can achieve *Overlap* in a distributed setting with only integrity assumptions on communication. This demonstrates that powerful (and expensive) mechanisms such as notaries are not necessary for all audit protocols.

The algorithms from Section 4 verify these statements for the special case when there are finitely many principals, the length of the provenance chain is bounded, and the internal state of the honest health care professional is bounded (i.e., they remember only a bounded number of messages). The extension of our methods to symbolic methods that permit handling infinite state spaces is left for future work.

6 Related work.

The security of the audit trail has built on advances in authenticated data structures (e.g., secure histories (Maniatis and Baker 2002), Persistent Authenticated Dictionaries (Anagnostopoulos et al. 2001) and Undeniable Attestations (Buldas et al. 2000)). This research has been used in specific applications. For example, PeerReview (Haeberlen et al. 2007) creates a per-node secure log, which records the messages a node has sent and received, and the inputs and outputs of the application. Node failures are detected by replaying such a trace against a reference implementation that is assumed to be determinate. CATS (Yumerefendi and Chase 2007) validates the integrity of storage hosted by a service provider. The clients are provided with the means to verify that all (and only) updates from authorized users are applied and seen. AudIt (Argyraki et al. 2007) is an explicit accountability interface for ISPs to supply feedback to traffic sources on QoS considerations. Accountability for the Internet protocol has also been investigated (Andersen et al. 2008). The APPLE system (Etalle and Winsborough 2007) suggests an architecture for a posteriori policy enforcement on documents: documents are always associated with policies, all clients operations on documents are logged, and distributed auditors occasionally verify the compliance with policies. In value-commitment protocols, a principal commits to a hidden value. Other principals cannot read this value, but can detect unlawful updates after the commitment. Fournet et al. (2008) studies such protocols using an applied pi-calculus.

These papers focus on efficient and expressive audit mechanisms to realize specific accountability policies. We study general models and limitations of accountability, aiming to provide a foundational analysis that can be incorporated as a component in the design of such systems. For example, the design goals of PeerReview include *Blamelessness* for honest agents and *Lower bound* for auditors. Our analysis provides a justification for the need to use secure ACKS to achieve these goals of PeerReview.

Cederquist et al. (2005) describe a policy language for data ownership and administrative issues. Cederquist et al. (2007) describe a system that uses audits to enforce

compliance to such policies. Proof-carrying-authorization forces the requestors of access to provide proofs validating their request. The AURA project (Vaughan et al. 2008) reuses these proofs for accountability via the "proofs as log entries" approach.

These papers focus on the design of logical methods to specify policies and enforce them via accountability. We study the design of the policies themselves, exploring the tradeoffs between the requirements that system policies place on honest agents and the power of audit protocols.

Our analysis methods are based on game-based logics for multiagent systems with *perfect* information, such as Alternating Temporal Logic (Alur et al. 2002) and coalition logics (see (Pauly 2001) for a historical survey). Such ideas have already been used to validate security protocols, e.g. (Mahimkar and Shmatikov 2005; Kremer and Raskin 2002).

7 Conclusions

We aim to develop foundations for distributed accountability systems. We have suggested an operational model and developed analysis methods using translations into games. Our running example suggests that our framework permits the designer of audit based accountability systems to explore the tradeoffs between the requirements on (a) the honest principals, (b) the guarantees provided by the communication network, and (c) the precision demanded of the audit protocol

Three important issues remain need to be addressed in future work: (a) the full integration with cryptographic primitives in the operational model, (b) quantitative models and methods such as Bloom filters are critical to achieving efficient audits of large datasets (Calandrino et al. 2007), and (c) equilibria notions provide an analysis of player intentions that is crucial to mechanism design.

Bibliography

M. Abadi, A. Birrell, and T. Wobber. Access control in a world of software diversity. In *Proc. of the Tenth workshop on Hot Topics in Operating Systems*, 2005. http://www.usenix.org/events/hotos05/.

R. Alur, T. Henzinger, and O. Kupferman. Alternating time temporal logic. *Journal of ACM*, 49:672–713, 2002.

A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applications. In *ISC*, volume 2200 of *LNCS*, pages 379–393. Springer, 2001.

D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *SIGCOMM*, pages 339–350. ACM, 2008.

K. Argyraki, P. Maniatis, O. Irzak, and S. Shenker. An accountability interface for the Internet. In *Proceedings of the 14th IEEE International Conference on Network Protocols*, 2007.

A. Barth, J. C. Mitchell, A. Datta, and S. Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294. IEEE Computer Society, 2007.

S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

A. Buldas, P. Laud, and H. Lipmaa. Accountable certificate management using undeniable attestations. In *ACM Conference on Computer and Communications Security*, pages 9–17, 2000.

J. A. Calandrino, J. A. Halderman, and E. W. Felten. Machine-assisted election auditing. In *EVT'07: Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, pages 9–9. USENIX Association, 2007.

J. G. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, and J. I. den Hartog. An audit logic for accountability. In *POLICY*, pages 34–43. IEEE Computer Society, 2005.

J. G. Cederquist, R. Corin, M. A. C. Dekker, S. Etalle, J. I. den Hartog, and G. Lenzini. Audit-based compliance control. *Int. J. Inf. Sec.*, 6(2-3):133–151, 2007.

A. Cirillo, R. Jagadeesan, C. Pitcher, and J. Riely. TAPIDO: Trust and authorization via provenance and integrity in distributed objects (extended abstract). In *Proceedings of ESOP 2008*, pages 208–223, 2008.

E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999. ISBN 0-262-03270-8.

S. Eriksén. Designing for accountability. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 177–186, 2002.

S. Etalle and W. H. Winsborough. A posteriori compliance control. In *SACMAT*, pages 11–20. ACM, 2007.

D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Computer Security Series. Artech House, 2003.

C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization policies. In *ESOP*, volume 3444 of *LNCS*, pages 141–156. Springer, 2005.

C. Fournet, N. Guts, and F. Z. Nardelli. A formal implementation of value commitment. In *Proceedings of ESOP 2008*, pages 383–397, 2008.

B. Friedman and J. Grudin. Trust and accountability: preserving human values in interactional experience. In *CHI '98: CHI 98 conference summary on Human factors in computing systems*, page 213. ACM, 1998.

P. Godefroid. Verisoft: A tool for the automatic analysis of concurrent reactive software. In *CAV*, volume 1254 of *LNCS*, pages 476–479. Springer, 1997.

A. Haeberlen, P. Kouznetsov, and P. Druschel. PeerReview: practical accountability for distributed systems. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 175–188. ACM, 2007.

J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. In *CSFW*, pages 118–130, 2003.

M. Hennessy and T. Regan. A process algebra for timed systems. *Inf. Comput.*, 117(2): 221–239, 1995.

S. Kremer and J.-F. Raskin. Game analysis of abuse-free contract signing. In *Proc. of the 15th IEEE Computer Security Foundations Workshop*, pages 206–220, 2002.

B. W. Lampson. Computer security in the real world. *IEEE Computer*, 37(6):37–46, 2004.

N. Li and J. C. Mitchell. A role-based trust-management framework. In *DISCEX (1)*, pages 201–. IEEE Computer Society, 2003.

N. A. Lynch. Input/output automata: Basic, timed, hybrid, probabilistic, dynamic, ... In *CONCUR*, volume 2761 of *LNCS*, pages 187–188. Springer, 2003.

A. Mahimkar and V. Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *CSFW*, pages 287–301. IEEE Computer Society, 2005.

P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In *USENIX Security Symposium*, pages 297–312. USENIX, 2002.

M. Pauly. *Logic for Social Software*. PhD thesis, Institute for Logic, Language and Computation, Universiteit van Amsterdam, 2001. ILLC Dissertation Series 2001-10.

F. B. Schneider. Enforceable security policies. *Information and System Security*, 3(1): 30–50, 2000.

J. A. Vaughan, L. Jia, K. Mazurak, and S. Zdancewic. Evidence-based audit. In *CSF*, pages 177–191. IEEE Computer Society, 2008.

D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman. Information accountability. Technical Report MIT-CSAIL-TR-2007-034, MIT, June 2007. http://hdl.handle.net/1721.1/37600.

A. R. Yumerefendi and J. S. Chase. Trust but verify: accountability for network services. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 37. ACM, 2004.

A. R. Yumerefendi and J. S. Chase. Strong accountability for network storage. *Trans. Storage*, 3(3):11, 2007.

A Constructions

A process P_i for p is output-deterministic when, for all states P, Q, P' of P_i, if $P \xrightarrow{\mathsf{p}\to\mathsf{q}:m} Q$ and $P \xrightarrow{\mathsf{p}\to\mathsf{q}:m} Q'$ then $Q = Q'$, i.e., output actions from p are deterministic. A process P_i for p is input-deterministic when, for all states P, Q, Q' of P_i, if $P \xrightarrow{\mathsf{q}\to\mathsf{p}:m} Q$ and $P \xrightarrow{\mathsf{q}\to\mathsf{p}:m} Q'$ then $Q = Q'$, i.e., input actions from p are deterministic.

The key steps in our compilation, performed in order, are: (a) Modeling of the dishonest agents, (b) Associate unique processes with each of the finitely many agents, (c) Add a scheduler agent, (d) Convert the LTS to a Kripke structure.

Constructing most general dishonest processes. In order to represent and detect dishonesty, for each dishonest agent, we construct a "most general dishonest" agent from the specification of their honest behavior. This construction is analogous to the "most general environment" used in software model checking (Godefroid 1997).

The standard subset construction constructs a (finite-state) input and output-deterministic process for p from a given (finite-state) process for p. A most general dishonest process for p is constructed from a deterministic finite state process for p as follows: consider a new state Q with the following transitions:

– For all states P in P_i, for all m, q such that there is no transition with label $\mathsf{p}\to\mathsf{q}:m$ from P, we add a transition $P \xrightarrow{\mathsf{p}\to\mathsf{q}:m} Q$. For all states P in P_i such that there is no transition with label σ from P, we add a transition $P \xrightarrow{\sigma} Q$.

– For all labels $\mathsf{p}\to\mathsf{q}:m$, there are transitions $Q \xrightarrow{\mathsf{p}\to\mathsf{q}:m} Q$. For all labels $\mathsf{q}\to\mathsf{p}:m$, there are transitions $Q \xrightarrow{\mathsf{q}\to\mathsf{p}:m} Q$. There is a transition $Q \xrightarrow{\sigma} Q$.

The Q state serves as detectors of dishonesty by p.

Process per agent. The auditor process is the one encoding the fixed auditor strategy (so it is input-deterministic and output-deterministic). The agents who are guaranteed to be honest are associated with the unique honest processes as given in the arena, and the agents who can be dishonest are associated with the "most general dishonest" agent constructed in the previous step. Construct the parallel composition of these processes. The resulting LTS has only output moves.

Adding a scheduler. This step resolves the nondeterminism between principals at states of an LTS (following the folklore "nondeterminism is modeled as an extra player"). Given a finite state process P for $\{\mathsf{p}_1,\ldots,\mathsf{p}_n\}$ with states S, we add a new principal to model a scheduler that chooses which principal can exercise an output action.

We use $\langle\rangle$ for the empty sequence and $tail(\cdot)$ to yield the tail of a non-empty sequence. We define an LTS over a set of states $\mathscr{S}$ given by:

$$\{0\}\times S \quad \cup\{(\langle k_1,\ldots,k_m\rangle,s) \mid (\forall 1\le i\le n)(\forall \mathsf{p}_{k_i})(\exists \mathsf{q},\mathsf{r},m)(\mathsf{p}_{k_i}=\mathsf{r}\wedge\ s \xrightarrow{\mathsf{r}\to\mathsf{q}:m})\}$$

The states in $\{0,\langle\rangle\}\times S$ are under control of scheduler whereas a state $(\langle k_1,\ldots,k_m\rangle,s)$ is under the control of p_{k_i}. The set of labeled transitions, with two additional new labels $sched, idle$, are determined as follows:

Start scheduler: If $s \xrightarrow{\mathsf{r}\to\mathsf{q}:m}$ for some $\mathsf{p}_{k_i}=\mathsf{r}$, then $(0,s) \xrightarrow{sched} (\langle k_1,\ldots,k_m\rangle,s)$.
Timeout: If $s \xrightarrow{\sigma} t$, then $(\langle k_1,\ldots k_m\rangle,s) \xrightarrow{idle} (tail(\langle k_1,\ldots k_m\rangle,s))$ and $(\langle\rangle,s) \xrightarrow{\sigma} (0,t)$.
Output: If $s \xrightarrow{\mathsf{r}\to\mathsf{q}:m} t$, then $(\langle \mathsf{r},\ldots\rangle,s) \xrightarrow{\mathsf{r}\to\mathsf{q}:m} (0,t)$

This translation preserves traces.

Proposition 11. *Consider a sequence homomorphism that maps labels $sched, idle$ to the empty sequence and is identity on all other labels. Then, the image of the set of traces from $(0,s)$ equals the set of traces from state s.*

The translation also ensures that given the choice of a move for each principal p_i at state s_i, all resulting possibilities can be explored by the scheduler from $(0, s_1 \parallel s_2 \ldots s_n)$. "Start Scheduler" initializes scheduler by choosing an ordering of the principals who have output moves at s. "Timeout" provides the transitions necessary to create a σ transition if all principals choose this option. "Output" provides the transitions necessary to perform actual output transitions of a player.

LTS to Kripke structures. This is standard by converting labels into propositions. Given a LTS with state set S and label set $\mathscr{M}$. Let $\mathscr{S} = \mathscr{M}\times S$. The set of transitions are determined as follows:

$$(m,s) \longrightarrow (m',t) \quad \Leftrightarrow \quad s \xrightarrow{m'} t$$

We also add new propositions $guilt_{\mathsf{p}}$ for each principal p. A state $s_1 \parallel s_2 \ldots s_n$ has propositions $guilt_{\mathsf{q}}$ if s_i is a state added by the construction of dishonest process for q, so the presence of $guilt_{\mathsf{q}}$ on a state along a path indicates dishonesty of q.