Consumable Credentials in Logic-Based Access Control

Lujo Bauer, Kevin D. Bowers, Frank Pfenning, and Michael K. Reiter

February 10, 2006
CMU-CyLab-06-002

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Consumable Credentials in Logic-Based Access Control

Lujo Bauer, Kevin D. Bowers, Frank Pfenning, and Michael K. Reiter

Carnegie Mellon University

Abstract. We present a framework to support *consumable* credentials in a logic-based distributed authorization system. Such credentials convey use-limited authority (e.g., to open a door once) or authority to utilize resources that are themselves limited (e.g., to spend money). We design a framework based on linear logic to enforce the consumption of credentials in a distributed system, and to protect credentials from nonproductive consumption as might result from misbehavior or failure. Finally, we give several usage examples in the framework, and evaluate the performance of our implementation for use in a ubiquitous computing deployment at our institution.

1 Introduction

The use of formal logics to model (e.g., [17, 37]) or implement (e.g., [9]) distributed access-control decision procedures provides assurance that access control is implemented correctly [2]. Such assurance is beneficial in light of the complex interactions that such systems are designed to accommodate, which may involve policies constructed in a decentralized way and that utilize delegations, roles and groups. Logic-based access-control systems typically express these policy elements in digitally signed credentials, and use the credentials as premises in a formal proof that a reference monitor's policy is satisfied by the credentials.

Despite significant attention to these systems in the last decade [40, 38, 4, 5], a natural form of access-control policy remains largely unexplored by this line of research: policies that use *consumable* authority that can be exercised only a limited number of times. Numerous types of authority are consumable, typically because the real-world resource affected by exercising the authority is itself consumable: e.g., the authority to spend money, or to sell theater tickets, would fall into this category. As a tangible example, a job hosting service might require a proof not only that a client's submitted job is safe to execute [43], but in addition that the client committed the required fee to the service to execute the job. Existing decentralized access-control frameworks, which express authority by way of digitally signed credentials that are easily copied, would permit a client to utilize the same credential (and hence the same funds) for executing multiple jobs.

In looking to extend prior work on access-control logics to support consumption of authority, one is quickly led to *linear logic*, a type of logic in which an

inference expends the premises that enabled it [31]. For example, a proof constructed in linear logic that a client's job is safe to execute, which is dependent on the client submitting payment, would consume the payment credentials. Once the credential is used in a proof, it is consumed, thus making it unavailable for use in future proofs. This accurately describes the corresponding real-world scenario: money, once withdrawn from an account and applied to a purchase, is spent and cannot be used again.

Using linear logic to model access-control systems is an interesting but relatively straightforward exercise. Recent work has conclusively argued, however, that *implementing* distributed access-control systems using logical frameworks provides a significantly greater level of assurance of the systems' correctness than merely *modeling* these systems using logic [6, 9]. This greater assurance is a product of bridging the gap between a system's specification (which can be easily modeled) and the system's implementation (which departs from the specification and therefore the model). To benefit from this greater assurance of correctness for an implemented access-control system, we need to develop a distributed linear-logic framework that we can use as the basis of this system.

This task is more complicated than building distributed frameworks that implement classical or intuitionistic logic, as has been done heretofore. In these previous frameworks and systems, as long as the appropriate credentials can be located, proofs can be created and verified on different nodes and at different times. In linear logic, however, a credential is transient, in that its use on one node causes it to become unavailable throughout the entire system. Hence, the task of implementing such a linear-logic based distributed system is more difficult.

In this paper we develop a decentralized logical framework for enforcing the consumption of credentials. Our framework is very flexible in that it permits the enforcement of arbitrary, even dynamically determined limits on the use of credentials. For example, in a Chinese wall policy [16], a client that accesses one resource is then precluded from accessing another for which the client, by virtue of accessing the first resource, now has a conflict of interest. Our framework could be used to implement this policy, consuming the client's credential for the second resource upon the client's use of a credential for the first.

The high-level strategy for enforcing credential consumption in our framework is to issue each consumable credential in such a manner that the credential's use requires the consent of another entity, its *ratifier*. The credential's ratifier, which is named in the credential itself, tracks the use of the credential and limits that use accordingly. Though this high-level approach is unsurprising, its conceptual simplicity is somewhat deceptive, due to several challenges that it raises.

1. In a setting where the steps of constructing a proof of authority and checking that proof are distinct [4], it is unclear what constitutes a credential *use* and thus the moment at which a credential should be consumed. One possibility is consuming the credential upon the assembly of a proof in which it is a premise. Another possibility is consuming it when a reference monitor checks the proof. As we will see in Section 3, neither of these alternatives

is satisfactory, and we propose a third alternative that, we argue, is more compelling.

2. For many types of consumable credentials, not only must the credential's consumption be enforced, but its availability must be protected against wasted "consumption". That is, a credential's consumption should not occur without the authorized party gaining the benefit of having used it. (A failure to ensure such availability would be particularly of concern for, e.g., authority to spend money.) In particular, if a credential is "used" during the construction of a proof, but the proof cannot be completed due to the lack of another permission, then the credential should not be consumed (since no authority was, in fact, exercised).[1] Our approach to dealing with availability in our framework draws on techniques from fair contract signing [11].
3. A hallmark of the proof-carrying authorization technique from which we start [4] is that a reference monitor need not be changed even if the application-specific logic in which proofs are constructed is changed. This benefit stems from the reference monitor's use of a checker for some base logic, in which other application-specific logics can be defined. In building a framework to enforce consumable credentials, we would like to retain this feature to the extent possible, extending the reference monitor's checker with constructs that are resilient to changes in the underlying application-specific logic.

To summarize the contributions of this paper: We discuss our approach to addressing the above issues, and detail the design and implementation of a framework that supports consumable credentials. We also outline several use cases for this framework, and evaluate the key facets of our implementation that affect its performance.

2 Related Work

The study of logics for access-control gained prominence with the work on the Taos operating system [3]. Since then, significant effort has been put into formulating formal languages and logics (e.g., [3, 5, 14, 40]) that can be used to describe a wide range of practical scenarios. Initially, the focus was on formulating logics that would be able to describe common abstractions such as roles, groups, and delegation without admitting any counter-intuitive behavior [2, 36, 35, 37]. In many cases, these logics were designed to model an implemented access-control system or policy-specification language [1, 3, 32, 33, 39]; the logics often included modality (to express the viewpoints of different actors), the law of the excluded middle, and some high-order features (typically, limited quantification over formulas). The usefulness of mechanically generated proofs (e.g., that access to

[1] Even a valid and complete proof may not be accepted, in which case we use extra-logical means to restore the availability of any consumed credentials. As in the case where a ticket holder is refused entry to a concert due to overcrowding, she should expect reimbursement. We do not formally model this compensation process in the logic, nor discuss it further in this paper.

a resource should be granted) led to various efforts to balance the decidability and expressiveness of access-control logics. These efforts resulted in various first-order classical logics, each of which would describe a comprehensive but not exhaustive set of useful access-control scenarios [5, 34, 38, 40], and more powerful higher-order logics that served as a tool for defining simpler, application-specific ones [4]. More recently, intuitionistic logics have been investigated as providing a more accurate model of the real world than classical logics [30]. An increasing amount of attention is spent on formally proving that particular access-control logics are sound, not only with respect to some abstract model, but also with respect to the reality the logics are intended to model [6, 30].

In this body of work on access-control logics, a credential is typically created by digitally signing a formula (e.g., Alice digitally signs that she is delegating her authority to Bob). Upon verification of the signature, the credential is represented as a predicate in the logic (e.g., Alice **signed** (...)), after which its use is unencumbered (i.e., the predicate can be used as a premise in arbitrarily many proofs and can no longer be made unavailable). This leads to some difficulty in modeling standard revocation and expiration. To overcome this deficiency, the logic is typically extended with mechanisms that allow for enforcement to occur outside the logic. One such frequently used mechanism is to issue short-lived certificates and use them to implement counter-signing of longer-lived certificates (e.g., Alice **signed** (*countersigner* **signed** (...) $\rightarrow$ ...)). Although sometimes adequate, these methods are limited. Once a revocable certificate has been counter-signed and deemed valid, the number and scope of its uses cannot be further controlled. Similarly, these methods make it difficult to reason about what may be true in the future or what may have been true in the past.

Though not previously researched in the context of logic-based access control, consumability has been extensively studied in applications such as electronic cash [21–23, 45, 48]. Preventing double-spending is an instance of our problem in which the rules regarding consumption are simple: money can be spent only once. As such, it is not surprising that our solution has certain elements in common with these proposals, notably the use of an online server (the *ratifier*) to enforce the consumption of a credential.[2] While the technique we developed can be used to implement an electronic payment system, that is by no means the only application of consumable credentials, nor is such an application meant to compete with work already done in electronic payment systems. The novelty of our approach is the integration of this technique with a logic-based access-control framework, and in implementing a general primitive for enforcing a range of consumption policies for arbitrary consumable resources.

[2] Detecting double-spenders does not require an online server [19, 24, 44, 50]. However, merely detecting the misuse of authority is not sufficient for access control more generally.

3 Preliminaries and Goals

In this section we describe the goals of our consumable credential system. To do so, we first present an illustrative access-control logic, discuss extending it with consumable credentials, and then describe what it means to have a system that implements it.

3.1 Logic-Based Access-Control

In this section we introduce a simple access-control logic, adapted from [8], that is illustrative and will serve as a running example through the paper. Our access-control logic is inhabited by terms and formulas. The terms denote principals and strings, which are the base types of our logic.

The **key**() constructor elevates strings representing public keys to the status of principals; **key**(`pubkey`) is the principal who owns the private key that corresponds to `pubkey`. Principals may want to refer to other principals or to create local name spaces—this gives rise to compound principals. We will write $\mathsf{Alice}.\texttt{secretary}$ to denote the principal whom Alice calls "secretary."

To affirm that a formula F is true, Alice signs it with her private key. The resulting sequence of bits will be represented by the formula $\texttt{pubkey}_{\texttt{Alice}}$ **signed** F. If it can be inferred that Alice believes F, using inference rules of the logic, then we write Alice **says** F.

To describe a resource that a client wants to access, we introduce the **action**() constructor. The first parameter to this constructor is a string that describes the desired action (e.g., "pay"). The second parameter is a list of qualifications of the desired action (e.g., $\langle \mathsf{Bob}, \$100 \rangle$). To allow for unique resource requests, the last parameter of the **action**() constructor is a nonce. A principal believes the formula **action**(*action*, *parameters*, *nonce*) if she thinks that it is OK to perform *action* during the session identified by *nonce*. We will usually omit the nonce in informal discussion and simply say **action**(*action*) or **action**(*action*, *parameters*).

Delegation is described with the **speaksfor** and **delegate** predicates.
Alice **speaksfor** Bob indicates that Bob has delegated to Alice his authority to make access-control decisions about any resource or action.
delegate(Bob, Alice, *action*) transfers to Alice only the authority to perform the particular action called *action*.

An *environment* Γ denotes a multiset of hypotheses, and the *judgment* $\Gamma \vdash F$ denotes that F can be proved from Γ. The following inference rules define how new judgments can be derived.

$$\frac{\Gamma \vdash \textit{pubkey}\ \textbf{signed}\ F}{\Gamma \vdash \textbf{key}(\textit{pubkey})\ \textbf{says}\ F}\ (\textsc{says-i}) \qquad \frac{\Gamma \vdash A\ \textbf{says}\ (A.S\ \textbf{says}\ F)}{\Gamma \vdash A.S\ \textbf{says}\ F}\ (\textsc{says-ln})$$

$$\frac{\Gamma \vdash A\ \textbf{says}\ (B\ \textbf{speaksfor}\ A) \quad \Gamma' \vdash B\ \textbf{says}\ F}{\Gamma, \Gamma' \vdash A\ \textbf{says}\ F}\ (\textsc{speaksfor-e})$$

$$\frac{\Gamma \vdash A \textbf{ says } (B \textbf{ speaksfor } A.S) \quad \Gamma' \vdash B \textbf{ says } F}{\Gamma, \Gamma' \vdash A.S \textbf{ says } F} \qquad (\textsc{speaksfor-e2})$$

$$\frac{\Gamma \vdash A \textbf{ says } (\textbf{delegate}(A, B, U)) \quad \Gamma' \vdash B \textbf{ says } (\textbf{action}(U, P, N))}{\Gamma, \Gamma' \vdash A \textbf{ says } (\textbf{action}(U, P, N))} \qquad (\textsc{delegate-e})$$

In addition, several standard rules describe how hypotheses from Γ can be used to prove that formulas F are true. These rules are typically implicit in the study of access-control logics.

$$\frac{}{\Gamma, F \vdash F} \ (\textsc{hyp}) \qquad \frac{\Gamma, F, F \vdash G}{\Gamma, F \vdash G} \ (\textsc{contr}) \qquad \frac{\Gamma \vdash G}{\Gamma, F \vdash G} \ (\textsc{weak})$$

The HYP rule states that if the proof goal is available as a hypothesis then it can be derived directly. In practice, each of the elements in environment Γ has the form *pubkey* **signed** F, and is the projection of a verified digitally signed certificate into the logical environment. The contraction and weakening rules formalize the idea that a hypothesis may be used multiple times (CONTR) or need not be used at all (WEAK). This is intuitively consonant with the nature of digitally signed credentials, which can be copied and used without limit.

For the purpose of illustrating how these rules for manipulating hypotheses allow us to use a single hypothesis twice, we introduce an additional rule.

$$\frac{\Gamma \vdash F_1 \qquad \Gamma' \vdash F_2}{\Gamma, \Gamma' \vdash F_1 \wedge F_2} \qquad (\textsc{and-i})$$

With the help of this rule, we can now construct the following derivation.

$$\frac{\dfrac{\dfrac{}{\Gamma, A \textbf{ signed } F \vdash A \textbf{ signed } F}\ (\textsc{hyp}) \qquad \dfrac{}{\Gamma', A \textbf{ signed } F \vdash A \textbf{ signed } F}\ (\textsc{hyp})}{\Gamma, \Gamma', A \textbf{ signed } F, A \textbf{ signed } F \vdash (A \textbf{ signed } F) \wedge (A \textbf{ signed } F)}\ (\textsc{and-i})}{\Gamma, \Gamma', A \textbf{ signed } F \vdash (A \textbf{ signed } F) \wedge (A \textbf{ signed } F)}\ (\textsc{contr})$$

Notice that by the penultimate step we managed to derive $(A \textbf{ signed } F) \wedge (A \textbf{ signed } F)$ only by assuming that we had two copies of A **signed** F as hypotheses. The final step, through the application of the CONTR rule, explicitly allows a single A **signed** F hypothesis to be sufficient.

3.2 A Linear Logic for Access-Control

In this section we augment our access-control logic to admit the use of "consumable" credentials.

The desire to strictly control the use of credentials is at odds with the CONTR and WEAK rules introduced in Section 3.1. To describe the use of consumable credentials, we borrow from linear logic the idea of a separate environment of "linear" hypotheses [20]. A judgement will now have two multisets of hypotheses:

the standard (reusable) hypotheses Γ, and the linear (consumable) hypotheses Δ. A judgement is now written as $\Gamma; \Delta \vdash F$. The difference between Γ and Δ is that contraction and weakening apply to Γ but not to Δ. The relevant rules change as follows.

$$\frac{}{\Gamma, F; \cdot \vdash F}\ (\textsc{hyp}) \qquad \frac{}{\Gamma; F \vdash F}\ (\textsc{hyp2})$$

$$\frac{\Gamma, F, F; \Delta \vdash G}{\Gamma, F; \Delta \vdash G}\ (\textsc{contr}) \qquad \frac{\Gamma; \Delta \vdash G}{\Gamma, F; \Delta \vdash G}\ (\textsc{weak})$$

As before, the CONTR and WEAK rules allow unmetered use of the reusable hypotheses Γ. However, the lack of corresponding rules about the restricted hypotheses Δ means that each hypothesis in Δ must be used exactly once. Hence, if a particular digital certificate represents a consumable credential, we will project the corresponding hypothesis into Δ, whereas Γ will be inhabited by the projections of unrestricted credentials. A multi-use consumable credential that is used several times in a proof will require that the appropriate number of copies be added to Δ. Notice that the derivation of $(A\ \mathbf{signed}\ F) \wedge (A\ \mathbf{signed}\ F)$ shown in the previous section will still work only if the $A\ \mathbf{signed}\ F$ hypothesis is in the unrestricted environment Γ. If it were in Δ, the CONTR rule used in the last step of the derivation could not be applied, and the derivation would fail, enforcing the principle that each hypothesis in Δ can be used only once.

3.3 Consuming Credentials

We would now like to consider how to utilize this linear access-control logic in the implementation of access control in a distributed system. In the access-control context, the hypotheses of a proof are credentials, and the proof shows that a policy (the proved formula) is satisfied by the credentials. The primary challenge introduced when this proof involves linear hypotheses is enforcing their consumption. Within the context of a single proof this is straightforward, as the reference monitor that is checking the proof can employ a linear proof checker, making sure that the HYP2 rule is used to derive F exactly the number of times that it is present as a hypothesis in Δ.

In the scenarios that motivate our study, however, consumption of F should not be limited to one proof, but rather should be global. In particular, these scenarios are populated by principals who issue credentials, generate proofs, and verify proofs that they have communicated to each other. A proof generated by one principal is typically sent to a second principal as part of a request to access a resource controlled by that principal. In these scenarios, we must prevent not only the profligate use of a particular consumable credential within a single proof, but also such a credential's use in arbitrarily many different proofs that may be created or verified by different principals.

This cannot be enforced through locally checking a proof alone; some distributed coordination must take place. More fundamentally, the moment of "use"

at which the credential should be "consumed" is a subtle design decision with significant ramifications. One possibility is to consume a credential when a proof containing it is verified by a reference monitor. However, this makes it impossible to determine whether a proof is valid or invalid by simple examination; rather, validity becomes a temporal notion. Another alternative would be to consume the credential during proof construction when the linear inference rule HYP2 is used. However, proof construction is a distributed search process that explores numerous potential paths for proving a result,[3] terminating when one of these paths succeeds [8]. Since most of the explored paths do not lead to successful proofs, consuming credentials upon each application of linear inference rules in this search process would quickly consume most credentials without any benefit being realized from them.

For these reasons, we reject both of these design options, and explore a third option in this paper. In this design, hypothesis consumption occurs as a step after the main search process for constructing a proof is completed, but before the proof is checked. Intuitively, the proving process prior to this step proceeded under the implicit assumption that the consumable credentials Δ used in the proof are valid. The last stage of the proving process is then to explicitly verify that the consumable credentials are in fact available and to mark their uses, and, if appropriate, render the credentials unavailable for future proofs. This transforms a potential proof of access (a proof of the formula F) into an actual proof (a proof of $\Box F$).

$$\frac{\Gamma; \Delta \vdash M : F \qquad \Gamma'; \cdot \vdash \mathbf{ratified}(\Delta, M, F)}{\cdot; \cdot \vdash \mathbf{box}(C, M) : \Box F} \qquad \text{(BOX-I)}$$

Here, M denotes the proof term that describes the derivation of F. The proof term is implicitly present in every judgement, but when we do not make use of it, we omit it for clarity. $\mathbf{box}(C, M)$ is the proof term that describes the derivation of $\Box F$; it encapsulates the digital certificates C that give rise to the environments Γ, Γ', and Δ.

Informally, we refer to the application of this rule as "boxing" the proof of F. A boxed proof may be checked on any host and arbitrarily many times. The key part of boxing F is demonstrating that the consumable credentials have been **ratified**. As we will see in Sections 4 and 5, **ratified** has both a logical and an extra-logical meaning, and is implemented partly by a distributed operation that enforces the use of the consumable credentials that gave rise to Δ. Since it may be possible to construct different proofs that prove the same goal, ratification is performed with respect to both a goal F and the corresponding proof term M. All proofs of access in our system are of goals of the form $\Box F$, and are verified with respect to hypotheses in Γ, Γ', and Δ. The certificates C that give rise to Γ, Γ', and Δ are packaged with the logical proof M of F to form the self-contained

[3] While the proof search problem is of central importance to such a system, research has been done in systems similar to ours yielding tractable solutions to the search problem which apply to our system as well [26].

proof term $\mathbf{box}(C, M)$, which can be verified in the absence of any hypotheses and without consulting any external credentials.

Suppose that each consumable credential δ is created with an allowed number of uses $\#\delta$. The main property required of boxing is captured by the following safety condition:

Bounded Use Let formulas $\Box F_1$, $\Box F_2$, ... be those formulas proved via the BOX-I rule, and let Δ_1, Δ_2, ... be the linear environments used in those applications of BOX-I. Then, the multiset $\bigcup_i \Delta_i$ contains at most $\#\delta$ instances of δ.

In addition to this bounded-use property, for boxing to be useful it also must be atomic, i.e., either boxing succeeds, in which case all hypotheses in Δ are used, or boxing fails and none of the hypotheses in Δ are used. This property and its implementation will be discussed in Section 5. Another interesting property is one that describes the idea that bounded resources have either been used or are still available for use; however, except in how it directly relates to atomicity, we do not discuss this further in the current paper.

4 A Distributed Framework for Linear Access-Control Logics

In this section we describe a general distributed framework that supports the use of linear access-control logics like the one described in Sections 3.2 and 3.3.

One of our main goals is to make our framework general, that is, capable of encoding not just our access-control logic but also other access-control logics with consumable credentials. A way to achieve this flexibility is to base the system on a more general logic which allows various application-specific logics to be expressed within it. This approach has been employed previously [4] using higher-order logic [25].

The first issue to be resolved is the choice of base logic for our system. Our access-control logic differs from previous ones in that it supports linear reasoning. We have several options in deciding on a base logic that can support this additional feature. One option is to follow previous work in using a standard higher-order logic, which has been shown capable of expressing linear logics [46]. However, because higher-order logic has a different native notion of judgment, such an encoding of linear logic may be inelegant. A second option is to use a logic that melds the features of linear and higher-order logic [42]. The third possibility, and the one that we choose, is to use a metalogic such as LLF [18]. In this case, we can either encode our linear access-control logic in LLF directly, or encode a linear higher-order logic in LLF and then encode our access-control logic in this intermediate logic. Each of these two options has advantages; what is relevant for the current work is that we have a base logic with standard linear-logic notions of judgment and consumption of hypotheses. Not coincidentally, these notions embody the rules described in Section 3.2 for manipulating hypotheses

in our linear access-control logic. Additionally, the base logic either supports directly or allows us to define standard connectives such as implication ($\rightarrow$, which in the context of linear logic is traditionally denoted by $\multimap$), conjunction ($\wedge$, traditionally denoted by $\otimes$), and quantification ($\forall, \exists$).

Having settled on a suitable base logic, the embedding of the operators (such as **delegate**) and rules (such as DELEGATE-E) of our application-specific logic into the base logic is straightforward, and similar to that of previous work [4]. Difficulties arise when we tackle the elements of our logic that have more than just a local meaning, i.e., elements whose meaning is defined with respect to the entire, distributed system. In particular, we must answer the following questions: (1) What exactly is a consumable credential? and (2) What is the logical and operational meaning of **ratified**?

Before we proceed to formally codify consumable credentials and ratifying, let us revisit their intuitive meanings, which are linked. Ratification is the step of generating a proof of access in which the consumption of consumable credentials used by the proof is globally enforced. Ratification transforms a potential proof into an actual proof through verifying that each of the consumable credentials is in fact still available to be used. Since every credential, including consumable ones, is issued by some principal, it seems reasonable for that principal, or his proxy, to be responsible for keeping track of whether or not the credential has been consumed. Ratification will therefore have to involve communicating with each of a set of principals that are responsible for the consumption-accounting of the certificates involved in the proof.

Clearly, there are extra-logical aspects to the implementation of ratification and consumable certificates. The immediate question, however, is whether ratification and consumable certificates need to be new primitives in the logic, or whether it is possible and useful to at least partially define them with respect to already-present primitives. In our quest to develop a general framework for resource-consuming access-control logics, we would prefer the latter.

A consumable credential is one whose ultimate validity is determined only at time of ratification. That is, the credential's potential meaning is from the outset sufficiently clear that we can reason using it and create almost-complete proofs; at the same time, a final step will be necessary for that meaning to be formally realized. These requirements are captured by the following definition.

$$A\ \mathbf{signed}_{A'}\ F \equiv A\ \mathbf{signed}\ (\forall M \forall G.\mathbf{consents}(A', F, M, G) \rightarrow F)$$

From the moment it is created by A, it is clear that this credential represents A authorizing the statement F. At the same time, it can be logically proved that A authorizes F only if it can first be demonstrated that a *ratifier* A' confirms that the credential is still valid (i.e., has not been consumed), by issuing a ratification credential ($\mathbf{consents}(A', F, M, G)$). To tie the use of a consumable credential to a particular proof, the ratification credential includes both the proof goal G and the proof M. To make it possible to delay this confirmation until the last step of proof construction, we extend our application-specific logic with the following rule.

$$\frac{\Gamma; \Delta \vdash \mathit{pubkey}\ \mathbf{signed}_A\ F}{\Gamma; \Delta \vdash \mathbf{key}(\mathit{pubkey})\ \mathbf{says}\ F} \qquad (\textsc{says-i2})$$

This rule allows us to reason with consumable credentials by temporarily assuming that they are valid. As we will see, the ratification that takes place during application of the BOX-I rule will require that we provide a ratification credential for each consumable credential in the proof. As discussed in Sections 3.1 and 3.2, the premise of the SAYS-I2 rule, *pubkey* $\mathbf{signed}_A\ F$, can be derived only if the corresponding hypothesis exists in the environment Δ.

The definition of the **consents** predicate varies between different ratification protocols; these are discussed in detail in Section 5. The parameters to **consents** include the statement, F, of the consumable credential, the proof goal G towards which the credential is being used, and the actual proof M of that goal. The latter pieces are necessary to tie the ratification of a credential to its use in a particular proof; otherwise, a credential could be ratified as part of one proof, then extracted from that proof and reused. In the most straightforward case, **consents** can be implemented as the simple credential,

$$\mathbf{consents}(A', F, M, G) \equiv A'\ \mathbf{signed}\ \langle F, M, G\rangle \qquad (1)$$

Ratification should be possible only if all uses of consumable credentials in a potential proof are valid. The implementation of this operation as a protocol carried out by different nodes of a distributed system is described in Section 5. This protocol will involve conveying to each ratifier the consumable credentials to be used and convincing them that all ratifiers involved in an execution of the protocol are ratifying credentials that are used within the same proof. These properties, then, we would also like to capture by our definition of what it means for the credentials Δ to be ratified.

$$\frac{}{\Gamma; \cdot \vdash \mathbf{ratified}(\cdot, M, G)} \qquad (\textsc{rat1})$$

$$\frac{\Gamma; \cdot \vdash \mathbf{consents}(A', F, M, G) \qquad \Gamma; \cdot \vdash \mathbf{ratified}(\Delta, M, G)}{\Gamma; \cdot \vdash \mathbf{ratified}((\Delta, A\ \mathbf{signed}_{A'}\ F), M, G)} \qquad (\textsc{rat2})$$

Informally, if a linear environment is empty $(\cdot)$ then it is considered ratified with respect to any proof M of a goal G (RAT1). Otherwise, ratification is defined recursively (RAT2): a linear environment $(\Delta, A\ \mathbf{signed}_{A'}\ F)$ is considered ratified if the consumable credential $A\ \mathbf{signed}_{A'}\ F$ has a corresponding ratification credential $(\mathbf{consents}(A', F, M, G))$, and if the remainder of the linear environment (Δ) can also be ratified. Since each ratification credential is issued with respect to the current proof and proof goal, each ratifier can inspect the proof before consenting to the use of a consumable credential within that proof. The ratifier can also count and record the number of uses of a consumable credential in the proof, and give or withhold its consent accordingly.

Given this, a proof of access is constructed as follows. First, a client Alice requests from Bob that he grant her access to a resource. Bob responds with the statement of the theorem Alice must prove; typically, the statement is of the form $\Box(\mathsf{Bob}\ \textbf{says}\ \textbf{action}(\mathit{action}))$. Alice proceeds to construct a proof of $\mathsf{Bob}\ \textbf{says}\ \textbf{action}(\mathit{action})$ using consumable credentials $\mathsf{Charlie}\ \textbf{signed}_{\mathtt{RCharlie}}\ F_1$ and $\mathsf{Danielle}\ \textbf{signed}_{\mathtt{RDanielle}}\ F_2$. Once Alice has completed this subproof, she contacts the ratifiers of Charlie's and Danielle's credentials, sending them the (not yet boxed) subproof of $\mathsf{Bob}\ \textbf{says}\ \textbf{action}(\mathit{action})$ and requesting that each ratify the credential for which it is responsible. Upon verifying that the credential submitted for ratification has not been consumed, each ratifier records the use of the consumable credential and produces the appropriate ratification credential. This enables Alice to prove that the multiset of consumable credentials has been ratified and thereby that $\Box(\mathsf{Bob}\ \textbf{says}\ \textbf{action}(\mathit{action}))$. Alice sends this proof to Bob, along with all the relevant consumable credentials and their ratification credentials. After verifying the proof, Bob grants Alice access to the desired resource.

Our implementation of consumable credentials uses on-line servers (the ratifiers) to validate credentials, which raises the question of whether the consumable credentials themselves could simply be issued immediately prior to the time they are needed. Such an approach, however, would prohibitively curtail the ability to reason a priori about consumable credentials during the construction of proofs. Our techniques are also related to countersigning; the advantage of our approach lies in that we carefully address what it means to consume multiple different credentials in the course of creating a single proof. This is done in such a way to prevent both the reuse of these credentials in other proofs and their needless consumption in the course of constructing proofs that will ultimately fail; this latter point is discussed further in the next section.

To help it determine whether or not to ratify a particular credential, a ratifier will typically keep state on a per-credential basis (e.g., the use count). Though this is an additional burden on the ratifier, it is no more than the burden that is typically placed on normal credential issuers. Additionally, in many cases the per-credential state will have to be kept only as long as the credential remains unconsumed and has not yet expired. Because of this, in the scenarios we envision, we expect the burden of keeping state to be light.

5 Atomicity

The outline of the ratification protocol described in Sections 3.3 and 4 overlooks the possibility that the protocol fails, either because a ratifier finds that the credential it is being asked to ratify is already consumed, or because a ratifier is unavailable. If this were to happen, some consumable credential might be marked as used by its ratifier, even though the proof of $\Box(\mathsf{Bob}\ \textbf{says}\ \textbf{action}(\mathit{action}))$, and thus accessing the resource, fails. To avoid this, in this section we refine the ratification protocol to implement the following property, in addition to Bounded Use:

Atomicity The ratification protocol for $\Box F$ is atomic, in that either the ratifier for each consumable credential $\delta \in \Delta$ records each of the uses of δ in the proof of $\Box F$—and in this case the proof of $\Box F$ succeeds—or none of the ratifiers records any such uses.

Recall that each ratifier produces a digitally signed ratification credential to ratify each use of the consumable credential for which it is responsible. Implementing the contribution of these digital signatures atomically for the goal F can be achieved by running a multiparty *contract-signing* protocol (e.g., [11, 29]) among the ratifiers for the consumable credentials used in the proof of F. Informally, a contract-signing protocol is one in which either all honest signing parties obtain a contract bearing *all* parties' signatures, or no one does. In our context, each ratifier participates in a contract-signing protocol with the other ratifiers to contribute its ratification credential. Each ratifier engages in the protocol only if the consumable credential for which it is responsible is not yet consumed, and registers a use of the credential if and only if the contract-signing protocol succeeds.

There are many contract-signing protocols that can achieve our requirements. That said, the particular protocol in use may require that we adjust the definition of $\mathbf{consents}(A', F, M, G)$ to accommodate the possible outputs of the protocol. In particular, deterministic contract signing protocols typically employ a trusted third party to settle disputes among the signers.[4] The trusted party generally has the power to either "force" a signature from a participant who has promised in previous rounds to sign the contract, or to terminate the protocol and ensure no one receives a signed contract. So-called "optimistic" protocols seek to avoid contacting the third party except in exceptional cases.

Such contract-signing protocols can be distinguished by whether or not the contract output by the protocol enables a verifier to determine if a party's signature was forced by the third party. If so, then the third party is *visible* in the protocol (e.g., [10]); if not, it is *invisible* (e.g., [29]). If the protocol ensures an invisible third party, then the definition of $\mathbf{consents}(A', F, M, G)$ need not separately accommodate runs in which the third party is consulted and runs in which it is not; e.g., the definition in (1) would suffice. However, if the third party is visible, then the definition of $\mathbf{consents}(A', F, M, G)$ would consist of two disjuncts, one for the case when the third party is not consulted and one for the case in which it is. This latter disjunct is protocol-dependent and so we do not detail the alternatives here, but formulating this disjunct is straightforward for the third-party-visible contract signing protocols of which we are aware.

Another issue in the use of a contract-signing protocol that employs a third party is the question of what third party to use. While this choice is orthogonal to our techniques, we caution the reader against using the prover in this role, i.e., the component requesting access and so applying the BOX-I rule in the context of assembling a proof. In most applications, this component would gain greater

[4] There are probabilistic protocols for performing contract signing that do not employ a trusted third party, but they have an error bound at least linear with the number of rounds [13].

authority (e.g., unlimited use of a consumable credential) by misbehaving in the role of the third party in the contract signing protocol. For this reason, better choices include utilizing the reference monitor that will check the proof, or alternatively implementing the third party using a multiparty implementation among the ratifiers themselves. This latter alternative requires an assumption that a majority of the ratifiers behave honestly, but in this case the contract-signing protocol can be particularly efficient [10].

6 Example

Using the concepts described in this paper it is easy to implement a number of applications that use consumable resources. Money is one of the easiest consumable resources to think about, and indeed these techniques can be used to develop a payment system within a logic-based access-control framework. While we are not proposing this example system as an alternative to iKP [12], SET [47], NetBill [49] and other electronic commerce protocols, it does serve to illustrate the expression and manipulation of consumable resources in a logic-based access-control framework.

As an example, imagine Alice walks into a store, fills her shopping cart with items and proceeds to check out. Instead of giving the clerk cash or a credit card, she instead presents him with a proof that the store will be given its money.

In this scenario, Bob, the store owner, is the reference monitor. He controls the items in his store, and will only release them once he has been given a proof of payment. Just as with credit card payments, Bob doesn't need the money immediately, but he needs to be convinced that when he later submits the proof Alice gave him to his bank, he will be paid.

When Alice approaches the counter and begins to check out, Bob issues her a challenge describing the proof of payment that she must produce.

$$\mathcal{G} = \Box\ \mathsf{ACH}\ \textbf{says}\ \textbf{action}(\mathit{pay}, \langle \mathsf{Bob}, \$100 \rangle, \mathit{nonce})$$

The challenge contains a nonce that is used to ensure freshness, enforce that the consumable credentials were boxed with respect to this proof, and also to serve as a transaction identifier. Since Bob cares chiefly that he is paid and not who will pay him, the challenge requires the payment to be authorized by the Automated Clearing House (ACH), a trusted authority that facilitates transfers between banks. Alice's task is now to construct a proof of payment. She starts the proving process by stating her willingness to pay Bob.

$$\mathcal{C}_0 = \mathsf{Alice}\ \textbf{signed}\ \textbf{action}(\mathit{pay}, \langle \mathsf{Bob}, \$100 \rangle, \mathit{nonce})$$

Alice must now demonstrate that there exists a chain of **delegate** and **speaksfor** relations from herself to the ACH. She has reason to believe such a chain exists because she has an account in good standing with a bank that has been certified by the ACH.

During proof generation Alice obtains the following credentials.

$$\mathcal{C}_1 = K_{\texttt{BankA}} \textbf{ signed } (\textbf{key}(\texttt{K}_{\texttt{Alice}}) \textbf{ speaksfor key}(\texttt{K}_{\texttt{BankA}}).\texttt{Alice})$$
$$\mathcal{C}_1 = K_{\texttt{ACH}_{\texttt{BC}}} \textbf{ signed } (\textbf{key}(\texttt{K}_{\texttt{BankA}}) \textbf{ speaksfor key}(\texttt{K}_{\texttt{ACH}}).\texttt{BC.BankA})$$
$$\mathcal{C}_3 = K_{\texttt{ACH}} \textbf{ signed delegate}(\textbf{key}(\texttt{K}_{\texttt{ACH}}), \textbf{key}(\texttt{K}_{\texttt{ACH}}).\texttt{BC}, \mathit{pay})$$
$$\mathcal{C}_4 = K_{\texttt{ACH}_{\texttt{BC}}} \textbf{ signed delegate}(\textbf{key}(\texttt{K}_{\texttt{ACH}}).\texttt{BC}, \textbf{key}(\texttt{K}_{\texttt{ACH}}).\texttt{BC.BankA}, \mathit{pay})$$

The first two credentials describe the **speaksfor** relationships between Alice and her bank and between her bank and the Bank Certifier (BC) of the ACH. Credentials $\mathcal{C}_3$ and $\mathcal{C}_4$ form a delegation chain from the ACH to its Bank Certifier, and from there to Alice's bank (BankA). Using these delegations, any *pay* statement made by BankA has the authority of being made by the ACH; in other words, BankA is accredited by the ACH.

Alice must now find a delegation statement allowing her to spend money from her account.

$$\mathcal{C}_5 = K_{\texttt{BankA}} \textbf{ signed}_{\texttt{K}_{\texttt{RBankA}}} \textbf{ delegate}(\textbf{key}(\texttt{K}_{\texttt{BankA}}), \textbf{key}(\texttt{K}_{\texttt{BankA}}).\texttt{Alice}, \mathit{pay})$$

This credential differs from the others in that it is consumable—Alice is allowed to withdraw money only while her account has a positive balance. With this credential, Alice can construct the subproof

$$\mathcal{M} : \mathsf{ACH} \textbf{ says action}(\mathit{pay}, \langle \mathsf{Bob}, \$100 \rangle, \mathit{nonce})$$

All that remains is to ratify credential $\mathcal{C}_5$ and invoke the BOX-I rule to finish the proof. To obtain the ratification credential for $\mathcal{C}_5$, Alice submits the proof to BankA's ratifier, RBankA, which is named in that credential. The ratifier deducts \$100 from Alice's account and transfers that money to the ACH. He also creates the following ratification credential.

$$\mathcal{C}_6 = K_{\texttt{RBankA}} \textbf{ signed } \langle \textbf{delegate}(\textbf{key}(\texttt{K}_{\texttt{BankA}}), \textbf{key}(\texttt{K}_{\texttt{BankA}}).\texttt{Alice}, \mathit{pay}), \mathcal{M}, \\ \textbf{goal}(\textbf{key}(\texttt{K}_{\texttt{ACH}}) \textbf{ says action}(\mathit{pay}, \langle \mathsf{Bob}, \$100 \rangle, \mathit{nonce})) \rangle$$

With this credential in hand, Alice can prove

$$\textbf{ratified}(\{\mathcal{C}_5\}, \mathsf{ACH} \textbf{ says action}(\mathit{pay}, \langle \mathsf{Bob}, \$100 \rangle, \mathit{nonce})$$

Applying the BOX-I rule she completes the proof

$$\Box\, \mathsf{ACH} \textbf{ says action}(\mathit{pay}, \langle \mathsf{Bob}, \$100 \rangle, \mathit{nonce})$$

and submits it to Bob for verification. Bob, convinced he will be paid for the items in Alice's cart, releases them to Alice. Bob will later show the proof to his bank, which in turn will hand it over to the ACH, which will actually transfer the funds to Bob's account. The Bank records the nonce in the statement of Bob's proof to prevent Bob from cashing the proof again. The full proof can be seen in Appendix A.2.

7 Implementation

At the time of this writing, we are in the process of deploying a distributed authorization framework called Grey [7] to control access to offices and other physical space on two floors (more than 30,000 square feet) of a new building at our institution. To support this, during building construction each door was equipped with an electric strike controlled by an embedded computer. A user exercises her authority to open a door via her smartphone, which connects to the embedded computer using Bluetooth, and receives a goal to prove (including a nonce). The smartphone utilizes a distributed proving system (similar to the one described in [8]) to generate the proof, possibly with help from other smartphones that hold necessary credentials, and ships this proof to the embedded computer in order to open the door. Our plans include deploying Grey-capable smartphones to roughly 100 building residents.

We have developed the linear logic framework presented here as a means to implement access-control policies that the Grey system presently cannot. This includes, for example, the ability to delegate authority to open an office once (see Appendix A.1). As we expand this testbed to include vending machines, the need for a distributed authorization system supporting consumable credentials (e.g., denoting money) will only grow.

We have completed a prototype implementation of a contract-signing protocol via which consumable credentials are ratified (see Section 5). In our prototype implementation, proofs of access are represented in the LolliMon language [41], which supports the linear connectives crucial for defining our consumable credentials. To verify the validity of proofs—including that each consumable credential in the environment Δ is used exactly once—ratifiers and reference monitors use a LolliMon interpreter as a proof checker. For the scenarios that we consider, proofs that depend on consumable credentials can be generated by a syntax-driven backward-chaining algorithm such as the one we have employed in Grey so far [8].

As discussed in Section 5, a ratifier is invoked with a proof for a formula G. If the proof is valid, the ratifier then engages in the contract-signing protocol to ratify the credentials for which it is responsible (assuming it consents to their use). As such, the contract-signing protocol and the verifying of proofs by the ratifiers account for the primary additional costs incurred during proof generation in a distributed proving system such as the one we use [8]. The LolliMon interpreter that we use for proof verification is sufficiently fast for the proofs we consider that it is not a bottleneck, and we do not discuss it further here.

The contract-signing protocol that we have implemented [29] offers strong properties that make it ideally suited for our system: it guarantees atomicity regardless of the number of ratifiers that fail or misbehave (provided that the trusted third party does not), and it implements an invisible third party T. To achieve these properties, however, the protocol utilizes significant machinery: the protocol running among n ratifiers involves $O(n^3)$ messages in $O(n^2)$ rounds. Each message is accompanied by an efficient noninteractive zero-

knowledge proof [15] regarding its contents, the details of which we omit. The cost of each zero-knowledge proof in the protocol is dominated by 9 modular exponentiations by the prover, and 12 by the verifier. Of most relevance here, however, is the form of the final contract signature (i.e., $\mathbf{consents}(A', F, M, G)$, where F is the content of the credential being ratified and M is a proof term describing the derivation of the proof goal G of the proof of access) by a ratifier A': this final contract signature is a zero-knowledge proof that an ElGamal ciphertext [27], if decrypted using T's private key, would yield a particular target plaintext.[5] This proof can be constructed either by the ratifier A' who created the ciphertext, because it knows the ephemeral key used in the encryption, or by the trusted third party T, because it knows the private key with which to decrypt the ciphertext; see Garay et al. [28] for details.

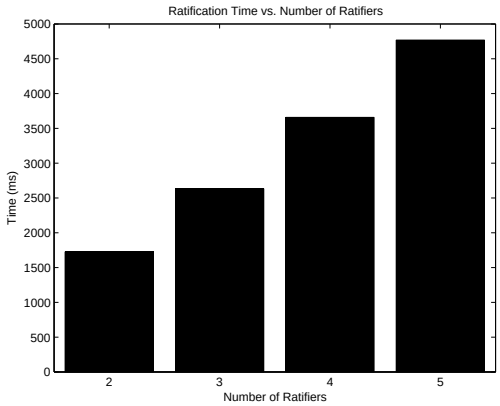

Fig. 1. The latency of the ratification protocol as a function of the number of ratifiers, measured from the standpoint of a client invoking the BOX-I rule in order to complete a proof of access.

The common-case latency (i.e., when the third party is not invoked) of our current prototype as a function of the number of participating ratifiers is shown in Figure 1. In these tests, each ratifier executed on a separate 2.8 GHz Pentium 4 computer. The latency of the ratification protocol is measured from the standpoint of a node invoking the BOX-I rule in order to complete a proof of access, and includes the cost of contacting all the ratifiers, carrying out the ratification protocol, and delivering ratification credentials to the client. A typical access-control proof involving consumable resources would likely depend on at

[5] If g is the generator of a multiplicative group $\mathcal{G}$ of prime order q, then the ElGamal ciphertext of $m \in \mathcal{G}$ under public key $y \in \mathcal{G}$ is (g^r, my^r), where $r \in \mathbb{Z}_q$ is a random "ephemeral key" generated anew for each encryption. The private key that enables decryption is $x = \log_g y$.

most two consumable credentials (and a greater number of reusable credentials), so the ratification cost for such a proof would be comparatively low; e.g., the sample proofs in Appendix A.1 and Appendix A.2 each make use of only a single consumable credential. A breakdown of the component costs of ratification as

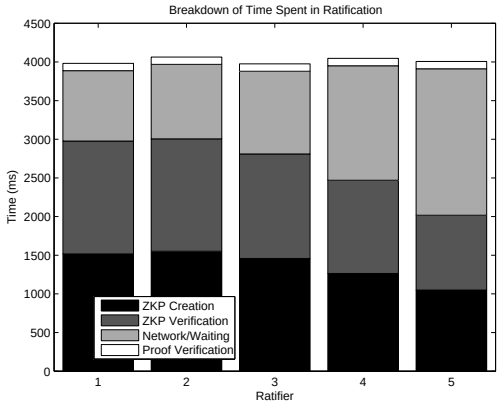

Fig. 2. Breakdown of costs involved in the ratification protocol for each of the five ratifiers participating in it.

measured on each of five ratifiers engaging in a contract-signing protocol is shown in Figure 2. The ratification protocol we implemented is asymmetric in that certain ratifiers create and verify more zero-knowledge proofs than other ratifiers; as a consequence, some ratifiers spend a majority of their time waiting to receive messages (Network/Waiting). Other major costs in the ratification protocol are generating the zero-knowledge proofs (ZKP Creation) communicated between the ratifiers, as well as verifying them (ZKP Verification). Note that the cost of the contract-signing protocol dominates the proof-checking time of the subgoal F: a proof of F containing 5 reusable and 5 linear credentials is verified by each ratifier in approximately 50 ms, with an additional 45 ms required to verify the validity of the digital certificates that the proof depends on.

The costs shown in Figure 1 are particularly pronounced because we implemented the prototype of our contract signing entirely in Java, for which each such modular exponentiation is significantly slower than an optimized native implementation. For this reason, we will soon enhance our implementation to perform these operations natively, and will report numbers for this implementation in the final version of the paper.

Acknowledgements

We gratefully acknowledge support from the National Science Foundation grant number CNS-0433540, the U.S. Navy grant number N00014-04-1-0724, and the U.S. Army Research Office contract number DAAD19-02-1-0389.

References

1. Martín Abadi. On SDSI's linked local name spaces. *Journal of Computer Security*, 6(1–2):3–21, October 1998.
2. Martín Abadi, Michael Burrows, Butler Lampson, and Gordon D. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.
3. Martín Abadi, Edward Wobber, Michael Burrows, and Butler Lampson. Authentication in the Taos Operating System. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, pages 256–269. ACM Press, December 1993.
4. Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*, pages 52–62, New York, NY, USA, 1999. ACM Press.
5. Dirk Balfanz, Drew Dean, and Mike Spreitzer. A security infrastructure for distributed java applications. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 15, Washington, DC, USA, 2000. IEEE Computer Society.
6. Lujo Bauer. *Access Control for the Web via Proof-carrying Authorization*. PhD thesis, Princeton University, November 2003.
7. Lujo Bauer, Scott Garriss, Jonathan M. McCune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the Grey system. In *Proceedings of the 8th Information Security Conference*, volume 3650, pages 431–445, September 2005.
8. Lujo Bauer, Scott Garriss, and Michael K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 81–95, Washington, DC, USA, May 2005. IEEE Computer Society.
9. Lujo Bauer, Michael A. Schneider, and Edward W. Felten. A general and flexible access-control system for the Web. In *Proceedings of the 11th USENIX Security Symposium*, pages 93–108, Berkeley, CA, USA, Aug 2002. USENIX Association.
10. Birgit Baum-Waidner. Optimistic asynchronous multi-party contract signing with reduced number of rounds. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, pages 898–911, London, UK, 2001.
11. Birgit Baum-Waidner and Michael Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In *ICALP*, pages 524–535, 2000.
12. M. Bellare, J.A. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. Van Herreweghen, and M. Waidner. Design, implementation, and deployment of the iKP secure electronic payment system. *IEEE Journal on Selected Areas in Communications*, 18(4):611–627, Apr 2000.
13. Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.
14. Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust-management system, version 2. Request For Comments (RFC) 2704, September 1999.

15. M. Blum, A. DeSantis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.
16. David F. C. Brewer and Michael J. Nash. The Chinese wall security policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 206–214, 1989.
17. Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.
18. Iliano Cervesato and Frank Pfenning. A linear logical framework. In E. Clarke, editor, *Proceedings of the Eleventh Annual Symposium on Logic in Computer Science — LICS'96*, pages 264–275, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press.
19. Agnes Hui Chan, Yair Frankel, and Yiannis Tsiounis. Easy come - easy go divisible cash. In *EUROCRYPT*, pages 561–575, 1998.
20. Bor-Yuh Evan Chang, Kaustuv Chaudhuri, and Frank Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Computer Science Department, Carnegie Mellon University, Apr 2003. Revised December 2003.
21. David Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology Proceedings of Crypto '82*, pages 199–203. Plenum, 1983.
22. David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.
23. David Chaum. Online cash checks. In *EUROCRYPT '89: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology*, pages 288–293, New York, NY, USA, 1990. Springer-Verlag New York, Inc.
24. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *CRYPTO '88: Proceedings on Advances in cryptology*, pages 319–327, New York, NY, USA, 1990. Springer-Verlag New York, Inc.
25. Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, pages 56–68, 1940.
26. Dwaine E. Clarke, Jean-Emile Elien, Carl M. Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.
27. Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Transactions on Information Theory*, 31:465–472, 1985.
28. Juan A. Garay, Markus Jakobsson, and Philip MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology – CRYPTO '99*, pages 449–466, 1999.
29. Juan A. Garay and Philip D. MacKenzie. Abuse-free multi-party contract signing. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 151–165, London, UK, 1999. Springer-Verlag.
30. Deepak Garg and Frank Pfenning. Non-interference in constructive authorization logic, October 2005. Submitted.
31. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
32. Joseph Y. Halpern and Ron van der Meyden. A logic for SDSI's linked local name spaces. In *CSFW '99: Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, page 111, Washington, DC, USA, June 1999. IEEE Computer Society.
33. Joseph Y. Halpern and Ron van der Meyden. A logical reconstruction of SPKI. In *CSFW '01: Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, page 59, Washington, DC, USA, 2001. IEEE Computer Society.

34. Joseph Y. Halpern and Vicky Weissman. Using first-order logic to reason about policies. In *Proceedings of the Computer Security Foundations Workshop*, page 187, June 2003.
35. Jon Howell. *Naming and sharing resources across administrative boundaries*. PhD thesis, Dartmouth College, May 2000.
36. Jon Howell and David Kotz. A formal semantics for SPKI. In *ESORICS '00: Proceedings of the 6th European Symposium on Research in Computer Security*, pages 140–158, London, UK, 2000. Springer-Verlag.
37. Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.
38. Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation logic: a logic-based approach to distributed authorization. *ACM Transactions on Information and Systems Security*, 6(1):128–171, February 2003.
39. Ninghui Li and John C. Mitchell. Understanding SPKI/SDSI using first-order logic. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 89–103, June 2003.
40. Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, Oakland, CA, May 2002.
41. Pablo López, Frank Pfenning, Jeff Polakow, and Kevin Watkins. Monadic concurrent linear logic programming. In *Proceedings of the 7th International Symposium on Principles and Practice of Declarative Programming*, pages 35–46, Lisbon, Portugal, July 2005.
42. Dale Miller. A multiple-conclusion meta-logic. *Theoretical Computer Science*, 165(1):201–232, 1996.
43. George C. Necula. Proof-carrying code. In *Proc. 24th ACM Symposium on Principles of Programming Languages, Paris, France*, pages 106–119. ACM Press, January 1997.
44. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *Crypto '91*, LNCS, pages 324–337. Springer-Verlag, 1992.
45. Birgit Pfitzmann and Michael Waidner. How to break and repair a "provably secure" untraceable payment system. In *Crypto '91*, LNCS 576, pages 338–350. Springer-Verlag, 1992.
46. Mehrnoosh Sadrzadeh. Modal linear logic in higher order logic, an experiment with Coq. Theorem Proving in Higher Order Logics, September 2003.
47. SET Secure Electronic Transaction LLC. *The SET Standard Specification*, May 1997.
48. Daniel R. Simon. Anonymous communication and anonymous cash. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 61–73, London, UK, 1996. Springer-Verlag.
49. M. Sirbu and J. D. Tygar. Netbill: An internet commerce system optimized for network delivered services. In *COMPCON '95: Proceedings of the 40th IEEE Computer Society International Conference*, page 20, Washington, DC, USA, 1995. IEEE Computer Society.
50. Hitesh Tewari, Donal O'Mahony, and Michael Peirce. Reusable off-line electronic cash using secret splitting. Technical report, Trinity College, 1998.

A Sample Proofs

A.1 One-Time Delegation

This example extends the current framework already in place at our institution (see Section 7). The two parties involved, Bob and Alice, both carry smartphones capable of generating proofs of access and communicating with the embedded doorend computers. Bob is a graduate student working for Alice. Alice is out of town, but Bob needs to get into her office to borrow a book. Alice would like to delegate to Bob the authority to open her door once, but only once. After Bob has used the delegation to open the door, he cannot use it again. In the current system, such a delegation is impossible.

In order to let Bob get into her office, (CIC-2525), Alice creates a consumable credential.

$$\mathcal{C}_0 = K_{\texttt{Alice}}\ \textbf{signed}_{\texttt{K}_{\texttt{RAlice}}}\ \textbf{delegate}(\textbf{key}(\texttt{K}_{\texttt{Alice}}), \textbf{key}(\texttt{K}_{\texttt{Bob}}), \mathit{CIC\ \ 2525})$$

Bob now walks up to Alice's door and asks it to open. The door responds with a challenge.

$$\mathcal{G} = \Box\ \mathsf{Alice}\ \textbf{says}\ \textbf{action}(\mathit{CIC\ \ 2525}, \langle \mathit{open} \rangle, \mathit{nonce})$$

The challenge includes a nonce, generated by the doorend computer, that will be used to ensure freshness of the response. Bob then generates the following credential.

$$\mathcal{C}_1 = K_{\texttt{Bob}}\ \textbf{signed}\ \textbf{action}(\mathit{CIC\ \ 2525}, \langle \mathit{open} \rangle, \mathit{nonce})$$

From $\mathcal{C}_0$ and $\mathcal{C}_1$, Bob can construct a proof of

$$\mathcal{M} : \mathsf{Alice}\ \textbf{says}\ \textbf{action}(\mathit{CIC\ \ 2525}, \langle \mathit{open} \rangle, \mathit{nonce})$$

which he submits for ratification. If this is the first time Bob has tried to use the delegation, his request will be ratified, and he will receive a ratification credential.

$$\begin{aligned}\mathcal{C}_2 = K_{\texttt{RAlice}}\ \textbf{signed}\ \langle &\textbf{delegate}(\textbf{key}(\texttt{K}_{\texttt{Alice}}), \textbf{key}(\texttt{K}_{\texttt{Bob}}), \mathit{CIC\ \ 2525}), \mathcal{M}, \\ &\textbf{goal}(\textbf{key}(\texttt{K}_{\texttt{Alice}})\ \textbf{says}\ \textbf{action}(\mathit{CIC\ \ 2525}, \langle \mathit{open} \rangle, \mathit{nonce}))\rangle\end{aligned}$$

The proof of

$$\textbf{ratified}(\{\mathcal{C}_0\}, \textbf{key}(\texttt{K}_{\texttt{Alice}})\ \textbf{says}\ \textbf{action}(\mathit{CIC\ \ 2525}, \langle \mathit{open} \rangle, \mathit{nonce}))$$

is straightforward, though slightly tedious, and will be omitted for readability. We will simply denote the proof of **ratified** by RATIFIED-I*, identifying the ratifying credential, ($\mathcal{C}_2$), that corresponds to the consumable credential, ($\mathcal{C}_0$), listed in the arguments. Bob then uses the BOX-I rule to complete the proof.

$$\mathcal{C}_0 = K_{\texttt{Alice}}\ \textbf{signed}_{\texttt{K}_{\texttt{RAlice}}}\ \textbf{delegate}(\textbf{key}(\texttt{K}_{\texttt{Alice}}), \textbf{key}(\texttt{K}_{\texttt{Bob}}), \textit{CIC 2525})$$
$$\mathcal{C}_1 = K_{\texttt{Bob}}\ \textbf{signed action}(\textit{CIC 2525}, \langle \textit{open} \rangle, \textit{nonce})$$
$$\mathcal{C}_2 = K_{\texttt{RAlice}}\ \textbf{signed}\ \langle \textbf{delegate}(\textbf{key}(\texttt{K}_{\texttt{Alice}}), \textbf{key}(\texttt{K}_{\texttt{Bob}}), \textit{CIC 2525}), \mathcal{M},$$
$$\textbf{goal}(\textbf{key}(\texttt{K}_{\texttt{Alice}})\ \textbf{says action}(\textit{CIC 2525}, \langle \textit{open} \rangle, \textit{nonce}))\rangle$$

1 $\textbf{key}(\texttt{K}_{\texttt{Alice}})\ \textbf{says delegate}(\textbf{key}(\texttt{K}_{\texttt{Alice}}), \textbf{key}(\texttt{K}_{\texttt{Bob}}), \textit{CIC 2525})$ SAYS-I2($\mathcal{C}_0$)

2 $\textbf{key}(\texttt{K}_{\texttt{Bob}})\ \textbf{says action}(\textit{CIC 2525}, \langle \textit{open} \rangle, \textit{nonce})$ SAYS-I($\mathcal{C}_1$)

3 $\textbf{key}(\texttt{K}_{\texttt{Alice}})\ \textbf{says action}(\textit{CIC 2525}, \langle \textit{open} \rangle, \textit{nonce})$ DELEGATE-E(1, 2)

4 $\textbf{ratified}(\{\mathcal{C}_0\}, \textbf{key}(\texttt{K}_{\texttt{Alice}})\ \textbf{says action}(\textit{CIC 2525}, \langle \textit{open} \rangle, \textit{nonce}))$ RATIFIED-I*($\mathcal{C}_2$)

5 $\Box\ \textbf{key}(\texttt{K}_{\texttt{Alice}})\ \textbf{says action}(\textit{CIC 2525}, \langle \textit{open} \rangle, \textit{nonce})$ BOX-I(3, 4)

A.2 Commerce

The following is an example proof of Alice paying Bob $100. In order for the payment to be accepted, the buyer (Alice) must generate a proof that the Automated Clearing House (ACH) **says action**(pay, ⟨Bob, $100⟩, $nonce$). The credentials necessary to complete the proof are $\mathcal{C}_0$–$\mathcal{C}_6$ below.

In order to initiate a purchase, the buyer (Alice) requests from the seller (Bob), the items in her shopping cart. Bob responds with a challenge to prove the boxed goal

$$\Box\mathsf{ACH}\ \textbf{says action}(\textit{pay}, \langle \mathsf{Bob}, \$100\rangle, \textit{nonce})$$

Bob generates the nonce to ensure freshness, enforce that boxing was performed with respect to this proof instance, and also to act as a transaction identifier. First, Alice generates credential $\mathcal{C}_0$. During proof generation, Alice obtains credential $\mathcal{C}_1$–$\mathcal{C}_5$. Using these she will generate a nearly complete proof, which she will submit to BankA's ratifier for him to generate the final necessary credential ($\mathcal{C}_6$). Once Alice has all of the credentials, she completes the proof and submits it to Bob. Bob will check the proof, and if successful, release the articles in Alice's shopping cart.

Credential $\mathcal{C}_0$, signifies Alice's willingness to pay Bob. Credentials $\mathcal{C}_1$, and $\mathcal{C}_2$ create **speaksfor** relationships between Alice and her bank, and between the bank and the Bank Certifier (BC) of the ACH. Credential $\mathcal{C}_3$ and $\mathcal{C}_4$ establish the delegation chain from the ACH through its Bank Certifier to Alice's bank (BankA), the authority to make pay statements.

Credential $\mathcal{C}_5$ is a consumable delegation from the bank to Alice. This credential requires ratification with respect to the proof

$$\mathcal{M} : \mathsf{ACH}\ \textbf{says action}(\textit{pay}, \langle \mathsf{Bob}, \$100\rangle, \textit{nonce})$$

at which point Alice will obtain ratification credential $\mathcal{C}_6$. Using this credential, it is straightforward but tedious to prove

$$\textbf{ratified}(\{\mathcal{C}_5\}, \mathsf{ACH}\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce))$$

We will omit the details for readability, and simply indicate the proof of **ratified** by RATIFIED-I*, noting the ratifying credentials that correspond to the consumable credentials listed as arguments.

Alice then invokes BOX-I, completing the proof.

$$\begin{aligned}
\mathcal{C}_0 &= K_{\mathtt{Alice}}\ \textbf{signed}\ (\textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce))\\
\mathcal{C}_1 &= K_{\mathtt{BankA}}\ \textbf{signed}\ (\textbf{key}(\mathtt{K}_{\mathtt{Alice}})\ \textbf{speaksfor}\ \textbf{key}(\mathtt{K}_{\mathtt{BankA}}).\mathtt{Alice})\\
\mathcal{C}_2 &= K_{\mathtt{ACH}_{\mathtt{BC}}}\ \textbf{signed}\ (\textbf{key}(\mathtt{K}_{\mathtt{BankA}})\ \textbf{speaksfor}\ \textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}.\mathtt{BankA})\\
\mathcal{C}_3 &= K_{\mathtt{ACH}}\ \textbf{signed}\ (\textbf{delegate}(\textbf{key}(\mathtt{K}_{\mathtt{ACH}}), \textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}, pay))\\
\mathcal{C}_4 &= K_{\mathtt{ACH}_{\mathtt{BC}}}\ \textbf{signed}\ (\textbf{delegate}(\textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}, \textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}.\mathtt{BankA}, pay))\\
\mathcal{C}_5 &= K_{\mathtt{BankA}}\ \textbf{signed}_{\mathtt{K}_{\mathtt{RBankA}}}\ (\textbf{delegate}(\textbf{key}(\mathtt{K}_{\mathtt{BankA}}), \textbf{key}(\mathtt{K}_{\mathtt{BankA}}).\mathtt{Alice}, pay))\\
\mathcal{C}_6 &= K_{\mathtt{RBankA}}\ \textbf{signed}\ \langle \textbf{delegate}(\textbf{key}(\mathtt{K}_{\mathtt{BankA}}), \textbf{key}(\mathtt{K}_{\mathtt{BankA}}).\mathtt{Alice}, pay), \mathcal{M},\\
&\qquad \textbf{goal}(\textbf{key}(\mathtt{K}_{\mathtt{ACH}})\textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce))\rangle
\end{aligned}$$

1	$\textbf{key}(\mathtt{K}_{\mathtt{BankA}})\ \textbf{says}\ \textbf{key}(\mathtt{K}_{\mathtt{Alice}})\ \textbf{speaksfor}\ \textbf{key}(\mathtt{K}_{\mathtt{BankA}}).\mathtt{Alice}$	SAYS-I($\mathcal{C}_1$)
2	$\textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}\ \textbf{says}\ \textbf{key}(\mathtt{K}_{\mathtt{BankA}})\ \textbf{speaksfor}\ \textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}.\mathtt{BankA}$	SAYS-I($\mathcal{C}_2$)
3	$\textbf{key}(\mathtt{K}_{\mathtt{ACH}})\ \textbf{says}\ \textbf{delegate}(\textbf{key}(\mathtt{K}_{\mathtt{ACH}}), \textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}, pay)$	SAYS-I($\mathcal{C}_3$)
4	$\textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}\ \textbf{says}\ \textbf{delegate}(\textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}, \textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}.\mathtt{BankA}, pay)$	SAYS-I($\mathcal{C}_4$)
5	$\textbf{key}(\mathtt{K}_{\mathtt{BankA}})\ \textbf{says}\ \textbf{delegate}(\textbf{key}(\mathtt{K}_{\mathtt{BankA}}), \textbf{key}(\mathtt{K}_{\mathtt{BankA}}).\mathtt{Alice}, pay)$	SAYS-I2($\mathcal{C}_5$)
6	$\textbf{key}(\mathtt{K}_{\mathtt{Alice}})\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)$	SAYS-I($\mathcal{C}_0$)
7	$\textbf{key}(\mathtt{K}_{\mathtt{BankA}}).\mathtt{Alice}\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)$	SPEAKSFOR-E2(6, 1)
8	$\textbf{key}(\mathtt{K}_{\mathtt{BankA}})\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)$	DELEGATE-E(7, 5)
9	$\textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}.\mathtt{BankA}\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)$	SPEAKSFOR-E2(8, 2)
10	$\textbf{key}(\mathtt{K}_{\mathtt{ACH}}).\mathtt{BC}\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)$	DELEGATE-E(9, 4)
11	$\textbf{key}(\mathtt{K}_{\mathtt{ACH}})\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)$	DELEGATE-E(10, 3)
12	$\textbf{ratified}(\{\mathcal{C}_5\}, \textbf{key}(\mathtt{K}_{\mathtt{ACH}})\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce))$	RATIFIED-I*($\mathcal{C}_6$)
13	$\Box\ \textbf{key}(\mathtt{K}_{\mathtt{ACH}})\ \textbf{says}\ \textbf{action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)$	BOX-I(11, 12)

A.3 Registration

Here we present an example utilizing consumable credentials in the framework of class registration. Consumable credentials are used both to limit the number of students signing up for a class, and to ensure that any class a student is trying to register for does not conflict with other classes he is already taking. This is done by modeling both the seats in a class and the timeslots in a weekly

schedule as consumable resources. Additionally, students are allowed to take only a limited number of credit hours in each semester. Each student must, therefore, also prove that by adding this class to their schedule, they will not surpass that limit. Assuming all these things are true, the student should be able to generate the necessary proof to register for a class.

In order to facilitate this proof, we augment our logic with the following inference rules.

$$\frac{\Gamma; \Delta \vdash F}{\Gamma; \Delta \vdash A \textbf{ says } F} \qquad \text{(SAYS-I3)}$$

$$\frac{\Gamma; \Delta \vdash A \textbf{ says } F \rightarrow G \quad \Gamma'; \Delta' \vdash A \textbf{ says } F}{\Gamma, \Gamma'; \Delta, \Delta' \vdash A \textbf{ says } G} \qquad \text{(SAYS-IMP-E)}$$

These rules are quite common in access-control logics and are compatible with the other rules we have described.

Alice wants to register for CS101, which meets on Monday, Wednesday, and Friday from 8:00 to 9:00 AM. To do so, she contacts the registrar, requesting that she be placed in the class. The registrar responds with a challenge:

$$\Box\ \mathsf{Registrar}\ \textbf{says}\ \textbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce})$$

When Alice's registration period began, she was given credentials $\mathcal{C}_0$–$\mathcal{C}_2$, along with similar ones for all weekly timeslots. The registrar, being in charge of seat assignments, also gave Alice credential $\mathcal{C}_3$. She obtained credential $\mathcal{C}_4$ during proof generation, and generated credential $\mathcal{C}_5$ herself. The registrar also gave her credential $\mathcal{C}_6$, specifying a subgoal Alice must prove before registration.

$$\begin{aligned}
\mathcal{C}_0 &= K_{\texttt{Calendar}}\ \textbf{signed}_{\mathrm{K_{RCal}}}\ \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Monday}, \mathit{0800{-}0900}\rangle)\\
\mathcal{C}_1 &= K_{\texttt{Calendar}}\ \textbf{signed}_{\mathrm{K_{RCal}}}\ \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Wednesday}, \mathit{0800{-}0900}\rangle)\\
\mathcal{C}_2 &= K_{\texttt{Calendar}}\ \textbf{signed}_{\mathrm{K_{RCal}}}\ \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Friday}, \mathit{0800{-}0900}\rangle)\\
\mathcal{C}_3 &= K_{\texttt{Registrar}}\ \textbf{signed}_{\mathrm{K_{RSeat}}}\ \textbf{action}(\mathit{seat}, \langle \mathit{F'05}, \mathit{CS101}\rangle, \mathit{nonce})\\
\mathcal{C}_4 &= K_{\texttt{Registrar}}\ \textbf{signed}_{\mathrm{K_{RCredit}}}\ \textbf{delegate}(\textbf{key}(\mathrm{K}_{\texttt{Registrar}}), \textbf{key}(\mathrm{K}_{\texttt{Alice}}), \mathit{credit_hours})\\
\mathcal{C}_5 &= K_{\texttt{Alice}}\ \textbf{signed action}(\mathit{credit_hours}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce})\\
\mathcal{C}_6 &= K_{\texttt{Registrar}}\ \textbf{signed}\ (\forall A.(\textbf{key}(\mathrm{K}_{\texttt{Calendar}})\ \textbf{says action}(\mathit{timeslot}, \langle A, \mathit{F'05}, \mathit{Monday}, \mathit{0800{-}0900}\rangle)\\
&\qquad \wedge\ \textbf{key}(\mathrm{K}_{\texttt{Calendar}})\ \textbf{says action}(\mathit{timeslot}, \langle A, \mathit{F'05}, \mathit{Wednesday}, \mathit{0800{-}0900}\rangle)\\
&\qquad \wedge\ \textbf{key}(\mathrm{K}_{\texttt{Calendar}})\ \textbf{says action}(\mathit{timeslot}, \langle A, \mathit{F'05}, \mathit{Friday}, \mathit{0800{-}0900}\rangle)\\
&\qquad \wedge\ \textbf{key}(\mathrm{K}_{\texttt{Registrar}})\ \textbf{says action}(\mathit{seat}, \langle \mathit{F'05}, \mathit{CS101}\rangle, \mathit{nonce})\\
&\qquad \wedge\ \textbf{key}(\mathrm{K}_{\texttt{Registrar}})\ \textbf{says action}(\mathit{credit_hours}, \langle A, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce}))\\
&\qquad \rightarrow \textbf{action}(\mathit{register}, \langle A, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce}))
\end{aligned}$$

This last credential can be thought of as requiring the student to be free on Monday, Wednesday, and Friday from 8:00 to 9:00 AM, a free seat to be available in the class, and the student to have 4 available credit hours for which they may

sign up. Note that we omit nonces from **action**() statements where they are unnecessary (e.g., $\mathcal{C}_0$–$\mathcal{C}_2$).

Alice now has enough credentials to prove

$$\mathcal{M} : \mathsf{Registrar}\ \textbf{says}\ \textbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce})$$

She then submits this proof for ratification of the consumable credentials it contains. Assuming all of the ratifiers consent to the use of their credentials, Alice will receive credentials $\mathcal{C}_7$–$\mathcal{C}_{11}$, allowing her to complete the proof.

$$\begin{aligned}
\mathcal{C}_7 &= K_{\mathtt{RCal}}\ \textbf{signed}\ \langle \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Monday}, \mathit{0800–0900}\rangle), \mathcal{M},\\
&\quad \textbf{goal}(\textbf{key}(\mathtt{K_{Registrar}})\textbf{says}\ \textbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce}))\rangle\\
\mathcal{C}_8 &= K_{\mathtt{RCal}}\ \textbf{signed}\ \langle \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Wednesday}, \mathit{0800–0900}\rangle), \mathcal{M},\\
&\quad \textbf{goal}(\textbf{key}(\mathtt{K_{Registrar}})\textbf{says}\ \textbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce}))\rangle\\
\mathcal{C}_9 &= K_{\mathtt{RCal}}\ \textbf{signed}\ \langle \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Friday}, \mathit{0800–0900}\rangle), \mathcal{M},\\
&\quad \textbf{goal}(\textbf{key}(\mathtt{K_{Registrar}})\textbf{says}\ \textbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce}))\rangle\\
\mathcal{C}_{10} &= K_{\mathtt{RSeat}}\ \textbf{signed}\ \langle \textbf{action}(\mathit{seat}, \langle \mathit{F'05}, \mathit{CS101}\rangle, \mathit{nonce}), \mathcal{M},\\
&\quad \textbf{goal}(\textbf{key}(\mathtt{K_{Registrar}})\textbf{says}\ \textbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce}))\rangle\\
\mathcal{C}_{11} &= K_{\mathtt{RCredit}}\ \textbf{signed}\ \langle \textbf{delegate}(\textbf{key}(\mathtt{K_{Registrar}}), \textbf{key}(\mathtt{K_{Student}}), \mathit{credit_hours}), \mathcal{M},\\
&\quad \textbf{goal}(\textbf{key}(\mathtt{K_{Registrar}})\textbf{says}\ \textbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce}))\rangle
\end{aligned}$$

Again we will omit the details of the proof of **ratified** and simply indicate ratifying credentials. We also combine steps as noted simply to save space.

$$\begin{array}{lll}
1 & \textbf{key}(\mathtt{K_{Calendar}})\ \textbf{says}\ \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Monday}, \mathit{0800–0900}\rangle) & \textsc{says-i2}(\mathcal{C}_0)\\
2 & \textbf{key}(\mathtt{K_{Calendar}})\ \textbf{says}\ \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Wednesday}, \mathit{0800–0900}\rangle) & \textsc{says-i2}(\mathcal{C}_1)\\
3 & \textbf{key}(\mathtt{K_{Calendar}})\ \textbf{says}\ \textbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{Friday}, \mathit{0800–0900}\rangle) & \textsc{says-i2}(\mathcal{C}_2)\\
4 & \textbf{key}(\mathtt{K_{Registrar}})\ \textbf{says}\ \textbf{action}(\mathit{seat}, \langle \mathit{F'05}, \mathit{CS101}\rangle, \mathit{nonce}) & \textsc{says-i2}(\mathcal{C}_3)\\
5 & \textbf{key}(\mathtt{K_{Registrar}})\ \textbf{says}\ \textbf{delegate}(\textbf{key}(\mathtt{K_{Registrar}}), \textbf{key}(\mathtt{K_{Alice}}), \mathit{credit_hours}) & \textsc{says-i2}(\mathcal{C}_4)\\
6 & \textbf{key}(\mathtt{K_{Alice}})\ \textbf{says}\ \textbf{action}(\mathit{credit_hours}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{4\ \ credits}\rangle, \mathit{nonce}) & \textsc{says-i}(\mathcal{C}_5)\\
7 & \textbf{key}(\mathtt{K_{Registrar}})\ \textbf{says}\ \textbf{action}(\mathit{credit_hours}, \langle \mathsf{Alice}, \mathit{F'05}, \mathit{4\ \ credits}\rangle, \mathit{nonce}) & \textsc{delegate-e}(5,6)\\
8 & \textbf{key}(\mathtt{K_{Registrar}})\ \textbf{says} & \textsc{says-i}(\mathcal{C}_6)\\
 & \quad (\forall A.(\textbf{key}(\mathtt{K_{Calendar}})\ \textbf{says}\ \textbf{action}(\mathit{timeslot}, \langle A, \mathit{F'05}, \mathit{Monday}, \mathit{0800–0900}\rangle) & \\
 & \qquad \wedge\ \textbf{key}(\mathtt{K_{Calendar}})\ \textbf{says}\ \textbf{action}(\mathit{timeslot}, \langle A, \mathit{F'05}, \mathit{Wednesday}, \mathit{0800–0900}\rangle) & \\
 & \qquad \wedge\ \textbf{key}(\mathtt{K_{Calendar}})\ \textbf{says}\ \textbf{action}(\mathit{timeslot}, \langle A, \mathit{F'05}, \mathit{Friday}, \mathit{0800–0900}\rangle) & \\
 & \qquad \wedge\ \textbf{key}(\mathtt{K_{Registrar}})\ \textbf{says}\ \textbf{action}(\mathit{seat}, \langle \mathit{F'05}, \mathit{CS101}\rangle, \mathit{nonce}) & \\
 & \qquad \wedge\ \textbf{key}(\mathtt{K_{Registrar}})\ \textbf{says}\ \textbf{action}(\mathit{credit_hours}, \langle A, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce})) & \\
 & \qquad \rightarrow \textbf{action}(\mathit{register}, \langle A, \mathit{CS101}, \mathit{F'05}, \mathit{4\ credits}\rangle, \mathit{nonce})) &
\end{array}$$

9 $(\forall A.(\mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle A, F'05, \mathit{Monday}, 0800\text{–}0900\rangle))$ IMP-I

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle A, F'05, \mathit{Wednesday}, 0800\text{–}0900\rangle))$ FORALL-E(8)

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle A, F'05, \mathit{Friday}, 0800\text{–}0900\rangle))$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{seat}, \langle F'05, \mathit{CS101}\rangle, \mathit{nonce})$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{credit\_hours}, \langle A, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce}))$

$\rightarrow \mathbf{action}(\mathit{register}, \langle A, \mathit{CS101}, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce}))$

$\rightarrow ((\mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Monday}, 0800\text{–}0900\rangle))$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Wednesday}, 0800\text{–}0900\rangle))$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Friday}, 0800\text{–}0900\rangle))$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{seat}, \langle F'05, \mathit{CS101}\rangle, \mathit{nonce})$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{credit\_hours}, \langle \mathsf{Alice}, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce}))$

$\rightarrow \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce}))$

10 $\mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}$ SAYS-I3

$((\mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Monday}, 0800\text{–}0900\rangle))$ SAYS-IMP-E(8, 9)

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Wednesday}, 0800\text{–}0900\rangle))$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Friday}, 0800\text{–}0900\rangle))$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{seat}, \langle F'05, \mathit{CS101}\rangle, \mathit{nonce})$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{credit\_hours}, \langle \mathsf{Alice}, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce}))$

$\rightarrow \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce}))$

11 $\mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Monday}, 0800\text{–}0900\rangle)$ AND-I($\times$4)(1–4, 7)

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Wednesday}, 0800\text{–}0900\rangle))$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Calendar}})\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Friday}, 0800\text{–}0900\rangle))$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{seat}, \langle F'05, \mathit{CS101}\rangle, \mathit{nonce})$

$\wedge\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{credit\_hours}, \langle \mathsf{Alice}, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce})$

12 $\mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce})$ SAYS-I3

SAYS-IMP-E(10, 11)

13 $\mathbf{ratified}(\{\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4\}, 12)$ RATIFIED-I*($\mathcal{C}_7$–$\mathcal{C}_{11}$)

14 $\Box\ \mathbf{key}(\mathtt{K}_{\mathtt{Registrar}})\ \mathbf{says}\ \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, 4\ \mathit{credits}\rangle, \mathit{nonce})$ BOX-I(12, 13)

Upon checking the proof, the registrar would then register Alice for CS101, as desired.