Consumable Credentials in Logic-Based Access-Control Systems

Kevin D. Bowers Lujo Bauer Deepak Garg Frank Pfenning Michael K. Reiter
Carnegie Mellon University
Pittsburgh, PA, USA
{kbowers, lbauer, dg, fp, reiter}@cmu.edu

Abstract

We present a method to implement 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., concert tickets). We design and implement mechanisms to enforce the consumption of credentials in a distributed system, and to protect credentials from nonproductive consumption as might result from misbehavior or failure. We explain how these mechanisms can be used to support a distributed authorization system that uses a linear access-control logic. 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., [18, 38]) or implement (e.g., [10]) 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 [41, 39, 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 [32]. 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, 10]. This greater assurance is a product of bridging the gap between a system's specification (which can be easily modeled) and its 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 tightly integrate linear logic with the basis of this distributed system.

This task is more complicated than building distributed systems around classical or intuitionistic logic, as has been done heretofore. In these previous 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 must cause 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 the mechanisms that permit a decentralized logic-based authorization system to enforce the consumption of credentials, and show how these mechanisms can be used to effectively enforce the abstractions of linear logic in a distributed setting. Our system is very flexible in that it permits the enforcement and straightforward specification of arbitrary, even dynamically determined limits on the use of credentials. For example, in a Chinese wall policy [17], 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. This policy can be specified in linear logic and enforced by our mechanism, which would consume the client's credential for the second resource upon the client's use of the first one.

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 until the authorized party commits to using 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). Our approach to dealing with availability draws on techniques from fair contract signing [12].

To summarize the contributions of this paper: We discuss our approach to addressing the above issues, and detail the design and implementation of a mechanism that allows a decentralized logic-based authorization system to support the consumption of credentials. For illustration, we present a linear access-control logic that uses this mechanism and show how it can encode several scenarios that make use of consumable credentials. We also empirically 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, 15, 41]) 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, 37, 38]. In many cases, these logics were designed to model an implemented access-control system or policy-specification language [1, 3, 33, 34, 40]; 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 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, 35, 39, 41], 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 closer tie between the policy (via formulas) and its enforcement (via proofs) [19, 31]. 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, 31].

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 $\mathbf{signed}$ $(\ldots)$), 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. Our ratification framework shares elements of this approach, though extends this idea to

tighter integration with logic-based authorization, enforcement of arbitrary consumption of credentials, and does so while preventing the capricious consumption of credentials.

Though not previously researched in the context of logic-based access control, consumability has been extensively studied in applications such as electronic cash [22, 23, 24, 45, 47]. 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.[1] 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 development of this technique within a logic-based access-control system, and in implementing a general primitive for enforcing a range of consumption policies for arbitrary consumable resources. An earlier version of this work can be found in [7].

3. Preliminaries and Goals

In this section we describe the goals of our consumable credential system. To be able to discuss the enforcement mechanism in concrete terms, 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

We introduce an illustrative logic which extends linear logic [32] with constructs for access control. It is based on [30]. The syntax of the logic contains terms and formulas. Terms are strings or principals (denoted by A), which are the base types in the logic. Generally, we refer to principals by name such as Alice or Bob. Principals may in turn want to refer to other principals or to create local name spaces which gives rise to compound principals. We will write Alice.secretary to denote the principal whom Alice calls "secretary."

To talk about a resource that a client wants to access, we introduce the $\mathbf{action}()$ predicate that takes three parameters as arguments. The first parameter to this predicate is a string that describes the desired action (e.g., "open"). The second parameter is a list of qualifications of the desired action (e.g., what should be opened). To allow for unique resource requests, the last parameter of the $\mathbf{action}()$ constructor is a nonce. The formula $\mathbf{action}(action, parameters, nonce)$ denotes 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 $\mathbf{action}(action, parameters)$ or $\mathbf{action}(action)$ if the qualifications are irrelevant. For simplicity, we assume that these are the only atomic formulas in the logic, but this is not a necessary restriction.

Our logic extends linear logic, which is a logic of resources. The primary judgment in linear logic is $F\ \mathbf{true}$ which means that there is exactly one copy of resource F. In order to model resources that may be used more than once, we use the judgment $F\ \mathbf{valid}$, which means that F may be used any number of times, including never. Logical reasoning is done using *hypothetical judgments* of the form $\Gamma; \Delta \vdash F\ \mathbf{true}$, where Γ and Δ are multisets of assumptions of the form $F\ \mathbf{valid}$ and $F\ \mathbf{true}$, respectively. The intuitive meaning of this judgment is that "by using each formula in Δ *exactly once* and using the formulas in Γ any number of times, one can obtain one copy of F".

This intuitive meaning is captured by the following rules of linear logic. The rule **hyp** says that we can prove F if Δ contains F and nothing else. Read bottom up, the rule **copy** says that we can copy resources from Γ into Δ as many times as required.

$$\frac{}{\Gamma; F\ \mathbf{true} \vdash F\ \mathbf{true}}(\mathbf{hyp})$$

$$\frac{\Gamma, F\ \mathbf{valid}; \Delta, F\ \mathbf{true} \vdash G\ \mathbf{true}}{\Gamma, F\ \mathbf{valid}; \Delta \vdash G\ \mathbf{true}}(\mathbf{copy})$$

In order to reason about access-control policies we need to be able to express the intent of a principal. This is represented using the judgment $A\ \mathbf{affirms}\ F$, which reads "principal A affirms the truth of formula F". Affirmation is different from truth because principals are not restricted in what they affirm. They may even affirm contradictory statements, without making the logic inconsistent. We assume that principals are rational in that they will not refuse to affirm a formula that is true. This gives us the following rule for affirmations.

$$\frac{\Gamma; \Delta \vdash F\ \mathbf{true}}{\Gamma; \Delta \vdash A\ \mathbf{affirms}\ F}(\mathbf{aff})$$

To be able to write affirmations inside formulas, we internalize $A\ \mathbf{affirms}\ F$ as a connective $A\ \mathbf{says}\ F$. This connective is defined by the following rules.

$$\frac{\Gamma; \Delta, F\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G}{\Gamma; \Delta, (A\ \mathbf{says}\ F)\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G}(\mathbf{saysL})$$

$$\frac{\Gamma; \Delta \vdash A\ \mathbf{affirms}\ F}{\Gamma; \Delta \vdash (A\ \mathbf{says}\ F)\ \mathbf{true}}(\mathbf{saysR})$$

[1] Merely *detecting* double-spenders does not require an online server [21, 25, 44, 49]. However, detecting the misuse of authority is not sufficient for access control more generally.

The **saysR** rule states that A **says** F is true whenever A **affirms** F. The **saysL** rule states that if we have an assumption $(A$ **says** $F)$ **true**, then we are justified in assuming A **true** while we are trying to prove some affirmation made by A. It turns out that we never need assumptions of the form A **affirms** F in Γ or Δ because these can immediately be replaced by assumptions $(A$ **says** $F)$ **true**.

While it is useful to reason about what Alice **says**, there are times when we would like to enforce that a statement actually came directly from Alice rather than as the conclusion of some other statements. This direct affirmation is written as Alice **signed** F. This can also be thought of as the sequence of bits resulting from Alice signing the formula F with her private key, using a standard cryptographic signature.

Since A **signed** F stands for direct evidence of A affirming F, it cannot be established by means of a proof. Hence it never occurs to the right of $\vdash$. There are two rules governing this judgment.

$$\frac{\Gamma; \Delta, F\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G}{\Gamma; \Delta, A\ \mathbf{signed}\ F \vdash A\ \mathbf{affirms}\ G}(\mathbf{signed})$$

$$\frac{\Gamma, A\ \mathbf{signed}\ F; \Delta, A\ \mathbf{signed}\ F \vdash G\ \mathbf{true}}{\Gamma, A\ \mathbf{signed}\ F; \Delta \vdash G\ \mathbf{true}}(\mathbf{copy}')$$

The rule **signed** is best thought of as saying that if A digitally signs the statement F, then we can assume that F is true while we are reasoning about an affirmation made by A. The second rule is similar to the rule **copy** and permits an indefinite number of copies of A **signed** F to be made if A **signed** F occurs in Γ.

Delegation is discussed in terms of 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 $action$. **delegate** and **speaksfor** can be defined in terms of other connectives as follows.

$$\mathbf{delegate}\ (A, B, U) \equiv \forall P.\forall N.(B\ \mathbf{says}\ \mathbf{action}(U, P, N) \longrightarrow A\ \mathbf{says}\ \mathbf{action}(U, P, N))$$

$$A\ \mathbf{speaksfor}\ B \equiv \forall U.\forall P.\forall N.(B\ \mathbf{says}\ \mathbf{action}(U, P, N) \longrightarrow A\ \mathbf{says}\ \mathbf{action}(U, P, N))$$

With these definitions we can derive the following rules which govern their use.

$$\frac{\Gamma; \Delta_1 \vdash B\ \mathbf{says}\ (\mathbf{action}(U, P, N))\ \mathbf{true} \quad \Gamma; \Delta_2, A\ \mathbf{says}\ (\mathbf{action}(U, P, N))\ \mathbf{true} \vdash F}{\Gamma; \Delta_1, \Delta_2, \mathbf{delegate}\ (A, B, U)\ \mathbf{true} \vdash F}(\mathbf{delegate})$$

$$\frac{\Gamma; \Delta_1 \vdash B\ \mathbf{says}\ (\mathbf{action}(U, P, N))\ \mathbf{true} \quad \Gamma; \Delta_2, A\ \mathbf{says}\ (\mathbf{action}(U, P, N))\ \mathbf{true} \vdash G}{\Gamma; \Delta_1, \Delta_2, (B\ \mathbf{speaksfor}\ A)\ \mathbf{true} \vdash G}(\mathbf{speaksfor})$$

The **speaksfor** rule states that if we can conclude that B **says** F, and assuming A **says** F we can conclude G, then assuming A **says** $(B$ **speaksfor** $A)$, we can also conclude G. The **delegate** rule is very similar, except the U variable must match in all three expressions.

3.2. Consuming Credentials

We would now like to consider how to utilize this linear access-control logic in a distributed system implementation. 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 consumable 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 which understands the distinction between environments and treats them appropriately.

In the scenarios that motivate our study, however, consumption of resources 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 **(hyp)** is used. However, proof construction is a distributed search process that explores numerous potential paths for proving a result,[2] terminating when one of these paths succeeds [9]. Since most of the explored paths do not lead to successful proofs, consuming credentials upon each application of

[2]The proof search process is a necessary ingredient for such a system, though since tractable, application-specific solutions to the search problem in such systems exists (e.g., [26]), we do not discuss it further here.

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 true. 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. We call this step *ratification*.

4. Ratification

Ratification is an extra-logical step which we use to enforce the linearity of our consumable credentials. Instead of the standard certificates, consumable credentials are created with respect to a ratifier that monitors their use and enforces their consumption. While the cryptographic signature does not differ between regular credentials and consumable credentials, we denote consumable credentials in the logic as $A\ \mathbf{signed}_{\mathsf{A}'}\ F$. The ratifier ($A'$) who is named in the logical representation, will later need to be contacted in order to ratify the consumable credential. Naturally there is also a new inference rule for dealing with such credentials.

$$\frac{\Gamma; \Delta, F\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G}{\Gamma; \Delta, A\ \mathbf{signed}_{\mathsf{A}'}\ F \vdash A\ \mathbf{affirms}\ G}(\mathbf{signed_L})$$

This rule can be used along the following lines of reasoning: if $A\ \mathbf{signed}_{\mathsf{A}'}\ F$ is available as a certificate, we can assume that F is $\mathbf{true}$ as long as we are reasoning about affirmations by A. Whether the resulting proof can actually be used in contingent upon the later ratification by A'.

4.1. Ratification Properties

There are two properties which must be enforced by our ratification mechanism. Suppose that each consumable credential δ is created with an allowed number of uses $\#\delta$. Then the following safety condition must hold.

Bounded Use Let formulas F_1, F_2, ... be those formulas proved in the system, and let Δ_1, Δ_2, ... be the linear environments used in those proofs. Then, the multiset $\bigcup_i \Delta_i$ contains at most $\#\delta$ instances of δ.

Informally, the system must enforce that the global number of uses of a consumable credential does not exceed the allowable uses as specified by the ratifier. How this is accomplished in a distributed setting will be discussed in more detail later.

While Bounded Use deals with bounding from above the number of uses of a consumable credential, we must also worry about ensuring the availability of valid consumable credentials. By this we mean to say that the system cannot waste consumable resources in a non-productive manner. This becomes immediately obvious in a system where consumable credentials are used to implement a form of currency. If money just disappeared out of your bank account because the system was able to waste resources, you would very quickly find a new system.

In a distributed proving environment, the risk of resource waste occurs after a proof has been completed, but before it has been ratified or checked. Suppose you construct a proof to purchase a ticket to the movies. The movie theater promised you a seat and you promised the money to pay for that seat. However, during ratification, your money is consumed, but there are no longer any seats available in the theater. Clearly, ratification must also enforce some sort of atomicity to ensure the previous scenario does not occur. This is captured in our second condition.

Atomicity The ratification protocol 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 F—and in this case the verification of F succeeds—or none of the ratifiers records any such uses.

Again, informally, the process of ratification must either occur for all credentials, or none of them. Either is an acceptable output from the system designer perspective, but there is no middle ground on which to stand.

4.2. Implementation

Bounded Use While ratification is an extra-logical mechanism, it is intrinsically tied to the logic. After the proof has been completed using consumable credentials, it must be sent to the applicable ratifiers who will certify that the consumable credentials are still valid. This is done by issuing ratification credentials which the ratifiers sign. These credentials are then appended to the reusable and consumable credentials gathered during proof search and sent to the reference monitor with the proof for verification.

Once the reference monitor receives the proof and corresponding digitally signed certificates, it first checks the cryptographic signatures on each credential. If the signatures are correct, it then populates both the reusable and linear environments, ensuring that Δ is only populated with consumable credentials if the corresponding ratification credentials are available.

Because of the Bounded Use requirement above, ratification credentials cannot be made with respect to only the credential they are ratifying. If that were the case, once a credential was ratified, the consumable credential and the

ratification credential could be copied and used in the construction of a later proof without contacting the ratifier to register another use. This is clearly unacceptable as the number of uses could not be controlled.

To overcome this, the ratification credential is not only created with respect to the consumable credential it is ratifying, but also with respect to the proof in which it is included. To this end, the proof generator, after completing the proof, sends the entire proof term, M, the proved formula F, and the credentials in both Γ and Δ to the ratifiers.

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. If the ratifier is willing to sign off on the uses of the consumable credential for which it is responsible it will issue a ratification credential with respect to both M and F.

The ratification credential then has the form $\langle \mathcal{C}, F, M \rangle_{A'}$, denoting the signature by A' on the consumable credential $\mathcal{C}$, the proof statement F, and the proof term M. Because the proof formula F contains a nonce, the returned ratification credential uniquely identifies a proof instance and cannot be reused, either in the same proof at a later time, or as a piece of a different proof.

Atomicity To deal with the issue of Atomicity, we borrow from work in contract signing. 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., [12, 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 the verifier know something about the protocol. In particular, deterministic contract signing protocols typically employ a trusted third party to settle disputes among the signers.[3] 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 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., [11]); if not, it is *invisible* (e.g., [29][4]). If the protocol ensures an invisible third party, then the verifier need not separately accommodate runs in which the third party is consulted and runs in which it is not. However, if the third party is visible, then the verifier must be willing to accept one of two possible signatures, 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.

An 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 in the context of assembling a proof. In most applications, this component would gain greater 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 [11].

Summary To summarize, 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 Bob $\mathbf{says}\ \mathbf{action}(action)$. Alice proceeds to construct a proof of Bob $\mathbf{says}\ \mathbf{action}(action)$ using consumable credentials Charlie $\mathbf{signed}_{\mathtt{RC}}\ F_1$ and Danielle $\mathbf{signed}_{\mathtt{RD}}\ F_2$. Once Alice has completed this proof, she contacts the ratifiers of Charlie's and Danielle's credentials, sending them the proof of Bob $\mathbf{says}\ \mathbf{action}(action)$ and requesting that each ratify the credential for which it is responsible. Upon verify-

[3] 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 in the number of rounds [14].

[4] This example invisible protocol has been shown to not be abuse free [20]: someone involved in the protocol can prove to an outsider that he has the power to force the the protocol to complete or abort. While undesirable in the case of general contract signing, abuse freedom is not necessary in the current application; we simply require atomicity here.

ing 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, which they send to Alice. Alice then sends the proof to Bob, along with all the relevant consumable credentials and their ratification credentials, which she just received. Bob then checks all of the cryptographic signatures and populates his reusable environment with the reusable credentials. For every consumable credential, Bob checks that a matching ratification credential is supplied and that the ratification credentials correspond to the current proof before adding the credential to his linear environment. Bob then verifies the proof and if successful, grants Alice access to the desired resource.

5. Discussion

Unsatisfied Requests The Atomicity property prevents capricious consumption of credentials during the proof process. However, once completed, even a valid and complete proof may not be accepted, in which case extra-logical means can be used to restore any consumed credentials. (As in the case where a ticket holder is refused entry to a baseball game due to a rainout, she should expect reimbursement.) To minimize the frequency of such occurrences and to aid their resolution when they do occur, the policy proved by a party requesting access could include a statement issued by the resource monitor indicating both the availability of the resource and that a valid proof will be rewarded with access. If this statement (itself a credential) is consumable when the resource is, proof construction would fail during ratification step if the resource monitor isn't able to promise delivery of the resource. Moreover, a ratified proof is evidence that can be presented to an arbiter in the event of a dispute.

Alternatives to Linear Logic Ratification is an extra-logical mechanism that we use to enforce the consumption of credentials, over and above the linear logic in which those credentials are expressed and used. This begs the question as to whether ratification or a similar mechanism should be modeled directly in the security logic, perhaps entirely avoiding the need to specify policies in linear logic, and instead permitting the use of a more standard or simpler logic.

We believe not, for two reasons. First, linear logic gives us a clean and intuitive abstraction for reasoning about consumable credentials. The specific details of how consumption is enforced are separate from the idea that a particular credential will be consumed after a fixed number of uses; hence, it is most straightforward for security policies to talk about the idea while remaining unencumbered by the details of the enforcement mechanism. This approach also makes it possible to have multiple enforcement mechanisms that all implement the same abstraction of consumable credentials.

Second, any enforcement mechanism that we pick must be able to enforce consumption not only in the distributed sense, but also within individual proofs (e.g., to prevent a consumable credential representing $10 from being used twice within the same proof). Although linear logic is not the only formalism that permits this sort of reasoning, any enforcement mechanism that we use will require similar logical machinery. Since linear logic has been widely studied and is well understood, there seems to be little benefit in using a different formalism. A similarly established type of logic that could instead be used to enforce credential consumption is *affine* logic. Both affine and linear logic expend hypotheses in Δ as they are used, but linear logic also requires that all the restricted hypotheses be used, whereas affine logic allows unused hypotheses in Δ to be discarded. Either type of logic would prevent over-consumption, but we chose linear logic to prevent the silent disappearance of consumable certificates in logical reasoning.

Alternatives to Consumable Credentials 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.

Ratifier Costs 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.

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 system as an alternative to iKP [13], SET [46], NetBill [48] 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.

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

The challenge contains a nonce that is used to ensure freshness, enforce that the consumable credentials were ratified 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.

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

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 four credentials.

$$\begin{aligned}
\mathcal{C}_1 &= \mathsf{BankA}\ \mathbf{signed}\ (\mathsf{Alice}\ \mathbf{speaksfor}\ \mathsf{BankA.Alice}) \\
\mathcal{C}_2 &= \mathsf{ACH.BC}\ \mathbf{signed}\ (\mathsf{BankA}\ \mathbf{speaksfor}\ \mathsf{ACH.BC.BankA}) \\
\mathcal{C}_3 &= \mathsf{ACH}\ \mathbf{signed}\ (\mathbf{delegate}\ (\mathsf{ACH}, \mathsf{ACH.BC}, pay)) \\
\mathcal{C}_4 &= \mathsf{ACH.BC}\ \mathbf{signed}\ (\mathbf{delegate}\ (\mathsf{ACH.BC}, \mathsf{ACH.BC.BankA}, pay))
\end{aligned}$$

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, i.e., BankA is accredited by the ACH.

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

$$\mathcal{C}_5 = \mathsf{BankA}\ \mathbf{signed}_{\mathsf{RBankA}}\ (\mathbf{delegate}\ (\mathsf{BankA}, \mathsf{BankA.Alice}, pay))$$

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 a proof of

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

All that remains is to ratify credential $\mathcal{C}_5$. 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.

$$\begin{aligned}
\mathcal{C}_6 = \mathsf{RBankA}\ \mathbf{signed}\ \langle &\mathbf{delegate}\ (\mathsf{BankA}, \mathsf{BankA.Alice}, pay), \mathcal{M}, \\
&\mathsf{ACH}\ \mathbf{says}\ \mathbf{action}(pay, \langle \mathsf{Bob}, \$100 \rangle, nonce) \rangle
\end{aligned}$$

With this credential in hand, Alice now has a ratified proof which she submits 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 B.2.

7. Implementation

At the time of this writing, we are in the process of deploying a distributed authorization framework called Grey [8] 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 [9]) 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 enforcement mechanism for linear logic presented here as a means to implement access-control policies that the current system presently cannot.

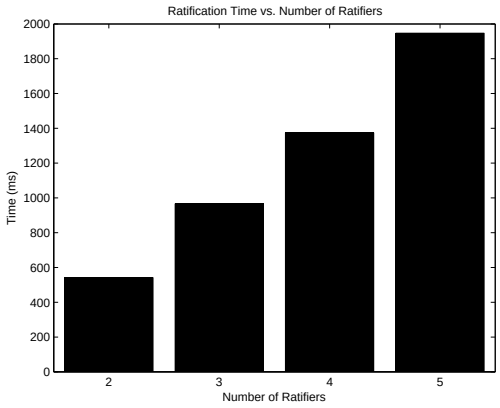

Figure 1. The latency of the ratification protocol as a function of the number of ratifiers, includes the cost of credential verification, proof checking, and the creation, verification and communication of non-interactive zero-knowledge proofs.

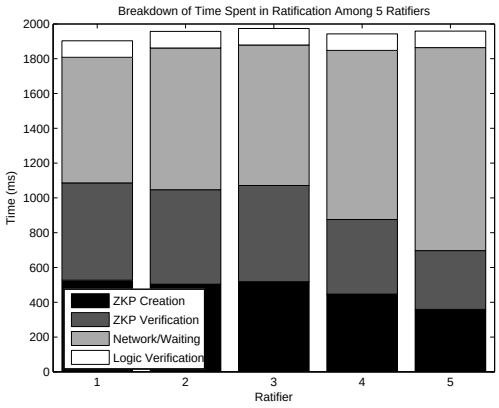

Figure 2. Breakdown of costs involved in the ratification protocol for each of the five ratifiers involved in a five-ratifier contract signing protocol.

This includes, for example, the ability to delegate authority to open an office once (see Appendix B.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 4). In our prototype implementation, proofs of access are represented in the LolliMon language [42], 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 logical proof checker. For the scenarios that we consider, proofs that depend on consumable credentials can be generated by a syntax-driven backward-chaining algorithm (e.g., [9]).

As discussed in Section 4, a ratifier is invoked with a proof term and a formula M and F. 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 [9]. 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.[5] 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 non-interactive zero-knowledge proof [16] 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. The form of the final contract signature by a ratifier A', 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, is a zero-knowledge proof that an ElGamal ciphertext [27], if decrypted using T's private key, would yield a particular target plaintext. This proof can be constructed either by the ratifier A' who created the ciphertext or by the trusted third party T; see Garay et al. [28] for details.

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 includes the cost of verifying the correctness of the submitted proof (including the digital signatures on the credentials contained therein) as well as the creation, verification and

[5] The protocol does not provide abuse freedom, but as noted earlier, this is not necessary in our use of the protocol.

communication among all ratifiers of the non-interactive zero-knowledge proofs used in the contract signing protocol. A typical access-control proof involving consumable resources would likely depend on at 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 B.1 and Appendix B.2 each make use of only a single consumable credential.

A breakdown of the component costs of ratification as 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.

The costs shown in Figure 1 are somewhat pronounced because we implemented the prototype of our contract signing primarily in Java, with only modular exponentiations optimized by a native implementation. Transition of the remaining cryptographic computations to native implementations should speed up the implementation even further.

Acknowledgments

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

References

[1] M. Abadi. On SDSI's linked local name spaces. *Journal of Computer Security*, 6(1–2):3–21, Oct. 1998.

[2] M. Abadi, M. Burrows, B. Lampson, and G. D. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, Sept. 1993.

[3] M. Abadi, E. Wobber, M. Burrows, and B. Lampson. Authentication in the Taos Operating System. In *Proceedings of the 14th ACM Symposium on Operating System Principles*, pages 256–269, Dec. 1993.

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

[5] D. Balfanz, D. Dean, and M. Spreitzer. A security infrastructure for distributed java applications. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 15–26, 2000.

[6] L. Bauer. *Access Control for the Web via Proof-carrying Authorization*. PhD thesis, Princeton University, Nov. 2003.

[7] L. Bauer, K. D. Bowers, F. Pfenning, and M. K. Reiter. Consumable credentials in logic-based access control. Technical Report CMU-CyLab-06-002, Carnegie Mellon University, Feb 2006.

[8] L. Bauer, S. Garriss, J. M. McCune, M. K. Reiter, J. Rouse, and P. Rutenbar. Device-enabled authorization in the Grey system. In *Proceedings of the 8th Information Security Conference*, pages 431–445, Sept. 2005.

[9] L. Bauer, S. Garriss, and M. K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 81–95, May 2005.

[10] L. Bauer, M. A. Schneider, and E. W. Felten. A general and flexible access-control system for the Web. In *Proceedings of the 11th USENIX Security Symposium*, pages 93–108, 2002.

[11] B. 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, 2001.

[12] B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 524–535, 2000.

[13] M. Bellare, J. Garay, R. Hauser, A. Herzberg, H. Krawczyk, M. Steiner, G. Tsudik, E. V. 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.

[14] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 36(1):40–46, 1990.

[15] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote trust-management system, version 2. Request For Comments (RFC) 2704, Sept. 1999.

[16] M. Blum, A. DeSantis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM Journal of Computing*, 20(6):1084–1118, 1991.

[17] D. F. C. Brewer and M. J. Nash. The Chinese wall security policy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[18] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[19] J. G. Cederquist, R. J. Corin, M. A. C. Dekker, S. Etalle, J. I. den Hartog, and G. Lenzini. The audit logic: Policy compliance in distributed systems. Technical Report TR-CTIT-06-33, Centre for Telematics and Information Technology, University of Twente, 2006.

[20] R. Chadha, S. Kramer, and A. Scedrov. Formal analysis of multi-party contract signing. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, pages 266–279, 2004.

[21] A. H. Chan, Y. Frankel, and Y. Tsiounis. Easy come - easy go divisible cash. In *Advances in Cryptology - Proceedings of Eurocrypt '98*, pages 561–575, 1998.

[22] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology, Proceedings of Crypto '82*, pages 199–203. 1983.

[23] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[24] D. Chaum. Online cash checks. In *Advances in Cryptology, Proceedings of Eurocrypt '89*, pages 288–293, 1990.

[25] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Advances in Cryptology, Proceedings of Crypto '88*, pages 319–327, 1990.

[26] D. E. Clarke, J.-E. Elien, C. M. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[27] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Transactions on Information Theory*, 31:465–472, 1985.

[28] J. A. Garay, M. Jakobsson, and P. MacKenzie. Abuse-free optimistic contract signing. In *Advances in Cryptology, Proceedings of Crypto '99*, pages 449–466, 1999.

[29] J. A. Garay and P. D. MacKenzie. Abuse-free multi-party contract signing. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 151–165, 1999.

[30] D. Garg, L. Bauer, K. D. Bowers, F. Pfenning, and M. K. Reiter. A linear logic of authorization and knowledge. In *Proceedings of the 11th European Symposium on Research in Computer Security*, pages 297–312, Sept. 2006.

[31] D. Garg and F. Pfenning. Non-interference in constructive authorization logic. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop*, pages 283–296, 2006.

[32] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[33] J. Y. Halpern and R. van der Meyden. A logic for SDSI's linked local name spaces. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 111–122, June 1999.

[34] J. Y. Halpern and R. van der Meyden. A logical reconstruction of SPKI. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, pages 59 – 70, 2001.

[35] J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 187 – 201, June 2003.

[36] J. Howell. *Naming and sharing resources across administrative boundaries*. PhD thesis, Dartmouth College, May 2000.

[37] J. Howell and D. Kotz. A formal semantics for SPKI. In *Proceedings of the 6th European Symposium on Research in Computer Security*, pages 140–158, 2000.

[38] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, Nov. 1992.

[39] N. Li, B. N. Grosof, and J. Feigenbaum. Delegation logic: a logic-based approach to distributed authorization. *ACM Transactions on Information and Systems Security*, 6(1):128–171, Feb. 2003.

[40] N. Li and J. C. Mitchell. Understanding SPKI/SDSI using first-order logic. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, pages 89–103, June 2003.

[41] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, May 2002.

[42] P. López, F. Pfenning, J. Polakow, and K. Watkins. Monadic concurrent linear logic programming. In *Proceedings of the 7th International Symposium on Principles and Practice of Declarative Programming*, pages 35–46, July 2005.

[43] G. C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, pages 106–119, Jan. 1997.

[44] T. Okamoto and K. Ohta. Universal electronic cash. In *Advances in Cryptoloty, Proceedings of Crypto '91*, pages 324–337. 1992.

[45] B. Pfitzmann and M. Waidner. How to break and repair a "provably secure" untraceable payment system. In *Advances in Cryptoloty, Proceedings of Crypto '91*, pages 338–350. 1992.

[46] SET Secure Electronic Transaction LLC. *The SET Standard Specification*, May 1997.

[47] D. R. Simon. Anonymous communication and anonymous cash. In *Advances in Cryptology, Proceedings of Crypto '96*, pages 61–73, 1996.

[48] M. Sirbu and J. D. Tygar. Netbill: An internet commerce system optimized for network delivered services. In *Proceedings of the 40th IEEE Computer Society International Conference*, pages 20 – 25, 1995.

[49] H. Tewari, D. O'Mahony, and M. Peirce. Reusable off-line electronic cash using secret splitting. Technical report, Trinity College, 1998.

Principals	A		
Propositions	F, G	$::=$	$\mathbf{action}(U, P, N) \mid A\ \mathbf{says}\ F \mid \ldots$
Categorical Judgments	J	$::=$	$F\ \mathbf{true} \mid F\ \mathbf{valid} \mid A\ \mathbf{affirms}\ F \mid A\ \mathbf{signed}\ F$
			$\mid A\ \mathbf{signed}_{\mathsf{A}'}\ F$
Unrestricted context	Γ	$::=$	$\cdot \mid \Gamma, F\ \mathbf{valid} \mid \Gamma, A\ \mathbf{signed}\ F$
Linear context	Δ	$::=$	$\cdot \mid \Delta, F\ \mathbf{true} \mid \Delta, A\ \mathbf{signed}_{\mathsf{A}'}\ F \mid \Delta, A\ \mathbf{signed}\ F$
Conclusions	γ	$::=$	$F\ \mathbf{true} \mid A\ \mathbf{affirms}\ F$
Hypothetical judgment			$\Gamma; \Delta \vdash \gamma$

$$\frac{}{\Gamma; F\ \mathbf{true} \vdash F\ \mathbf{true}}(\mathbf{hyp}) \qquad \frac{\Gamma, F\ \mathbf{valid}; \Delta, F\ \mathbf{true} \vdash G\ \mathbf{true}}{\Gamma, F\ \mathbf{valid}; \Delta \vdash G\ \mathbf{true}}(\mathbf{copy})$$

$$\frac{\Gamma, A\ \mathbf{signed}\ F; \Delta, A\ \mathbf{signed}\ F \vdash G\ \mathbf{true}}{\Gamma, A\ \mathbf{signed}\ F; \Delta \vdash G\ \mathbf{true}}(\mathbf{copy}') \qquad \frac{\Gamma; \Delta \vdash F\ \mathbf{true}}{\Gamma; \Delta \vdash A\ \mathbf{affirms}\ F}(\mathbf{aff})$$

$$\frac{\Gamma; \Delta \vdash A\ \mathbf{affirms}\ F}{\Gamma; \Delta \vdash (A\ \mathbf{says}\ F)\ \mathbf{true}}(\mathbf{saysR}) \qquad \frac{\Gamma; \Delta, F\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G}{\Gamma; \Delta, (A\ \mathbf{says}\ F)\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G}(\mathbf{saysL})$$

$$\frac{\Gamma; \Delta, F\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G}{\Gamma; \Delta, A\ \mathbf{signed}\ F \vdash A\ \mathbf{affirms}\ G}(\mathbf{signed}) \qquad \frac{\Gamma; \Delta, F\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G}{\Gamma; \Delta, A\ \mathbf{signed}_{\mathsf{A}'}\ F \vdash A\ \mathbf{affirms}\ G}(\mathbf{signed_L})$$

Figure 3. Summary of the logic

A. Summary of the Logic

This section summarizes our logic and a cut-elimination theorem for it. See [30] for details of a closely related logic. The standard connectives have been omitted to save space.

Theorem 1 (Admissibility of cut). *The following hold in the logic described above.*

1. *If* $\Gamma; \Delta_1 \vdash A\ \mathbf{true}$ *and* $\Gamma; \Delta_2, F\ \mathbf{true} \vdash \gamma$*, then* $\Gamma; \Delta_1\Delta_2 \vdash \gamma$.
2. *If* $\Gamma; \cdot \vdash F\ \mathbf{true}$ *and* $\Gamma, F\ \mathbf{valid}; \Delta \vdash \gamma$*, then* $\Gamma; \Delta \vdash \gamma$.
3. *If* $\Gamma; \Delta_1 \vdash A\ \mathbf{affirms}\ F$ *and* $\Gamma; \Delta_2, F\ \mathbf{true} \vdash A\ \mathbf{affirms}\ G$*, then* $\Gamma; \Delta_1\Delta_2 \vdash A\ \mathbf{affirms}\ G$.

Proof. By nested induction, first on the size of the cut formula and then on the size of the two given derivations. □

B. Sample Proofs

B.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.

$$\mathcal{C}_0 = \mathsf{Alice}\ \mathbf{signed}_{\mathsf{RAlice}}\mathbf{delegate}\ (\mathsf{Alice}, \mathsf{Bob}, \mathit{CIC\ 2525})$$

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

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

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.

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

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

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

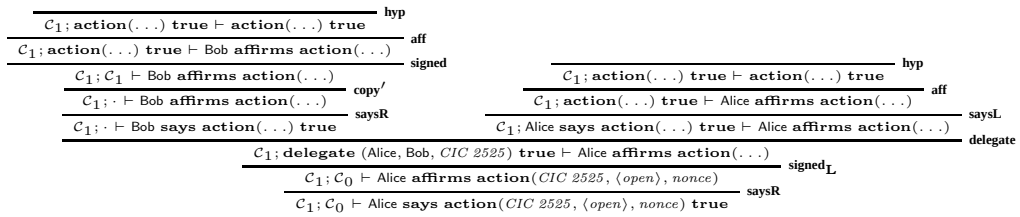

Figure 4. Proof of Alice says action(...) true

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.

$$\mathcal{C}_2 = \mathsf{RAlice}\ \mathbf{signed}(\mathbf{delegate}\ (\mathsf{Alice}, \mathsf{Bob}, \mathit{CIC\ 2525}), \mathcal{M}, \mathsf{Alice}\ \mathbf{says}\ \mathbf{action}(\mathit{CIC\ 2525}, \langle \mathit{open} \rangle, \mathit{nonce}))$$

Bob can then append this credential to the others he collected during proof construction and submit them, along with the proof, to the doorend computer. The computer, after checking the signatures and populating its environments will check that the proof goal is in fact satisfied by the credentials submitted, and if so will open the door for Bob.

$$\begin{aligned}
\mathcal{C}_0 &= \mathsf{Alice}\ \mathbf{signed}_{\mathsf{RAlice}}\mathbf{delegate}\ (\mathsf{Alice}, \mathsf{Bob}, \mathit{CIC\ 2525}) \\
\mathcal{C}_1 &= \mathsf{Bob}\ \mathbf{signed}\ \mathbf{action}(\mathit{CIC\ 2525}, \langle \mathit{open} \rangle, \mathit{nonce}) \\
\mathcal{C}_2 &= \mathsf{RAlice}\ \mathbf{signed}\ \langle \mathbf{delegate}\ (\mathsf{Alice}, \mathsf{Bob}, \mathit{CIC\ 2525}), \mathcal{M}, \\
&\qquad \mathsf{Alice}\ \mathbf{says}\ \mathbf{action}(\mathit{CIC\ 2525}, \langle \mathit{open} \rangle, \mathit{nonce}) \rangle
\end{aligned}$$

A simplified version of the proof derivation is presented in Figure 4. It is easiest read from bottom up.

B.2. Commerce

The following is an example proof of Alice paying Bob $100. In order for the payment to be accepted, Alice must generate a proof that the Automated Clearing House (ACH) $\mathbf{says}\ \mathbf{action}(\mathit{pay}, \langle \mathsf{Bob}, \$100 \rangle, \mathit{nonce})$.

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 goal

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

Bob generates the nonce to ensure freshness, ensure that the ratification credentials are issued with respect to this proof, and also to act as a transaction identifier. First, Alice generates credential $\mathcal{C}_0$. During proof generation, Alice obtains credentials $\mathcal{C}_1$–$\mathcal{C}_5$. Using these she will generate a 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 submits the proof and credentials to Bob. Bob will verify that the credentials are valid, and check that all consumable credentials have ratification credentials and, after populating his environments, that the proof is correct. If the verification succeeds, Bob will release the articles in Alice's shopping cart.

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

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 of

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

after which Alice will get ratification credential $\mathcal{C}_6$, which she submits along with the proof to Bob for verification.

Again, the proof in Figure 5 is easiest to read from the bottom up, starting with the unlabeled proof tree.

B.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

$\mathcal{D}_4$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathbf{action}(\ldots)\ \mathbf{true}}\ \textbf{hyp}}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathsf{Alice}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{aff}}{\mathcal{C}_{0-4};\mathcal{C}_0 \vdash \mathsf{Alice}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{signed}}{\mathcal{C}_{0-4};\mathcal{C}_0 \vdash \mathsf{Alice}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{saysR}}{\mathcal{C}_{0-4};\cdot \vdash \mathsf{Alice}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{copy}'$$

$$\dfrac{\dfrac{\dfrac{}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathbf{action}(\ldots)\ \mathbf{true}}\ \textbf{hyp}}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathsf{BankA.Alice}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{aff}}{\mathcal{C}_{0-4};\mathsf{BankA.Alice}\ \mathbf{says\ action}(\ldots)\ \mathbf{true} \vdash \mathsf{BankA.Alice}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{saysL}$$

$$\dfrac{\cdots}{\mathcal{C}_{0-4};\mathcal{A}_1 \vdash \mathsf{BankA.Alice}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{speaksfor}$$

$\mathcal{D}_3$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_4}{\mathcal{C}_{0-4};\mathcal{A}_1 \vdash \mathsf{BankA.Alice}\ \mathbf{affirms\ action}(\ldots)}}{\mathcal{C}_{0-4};\mathcal{A}_1 \vdash \mathsf{BankA.Alice}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{saysR}}{\mathcal{C}_{0-4};\mathcal{A}_1, \mathbf{delegate}\ (\mathsf{BankA}, \mathsf{BankA.Alice}, pay)\ \mathbf{true} \vdash \mathsf{BankA}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{signed}_{\mathbf{L}}}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{A}_1 \equiv (\mathsf{Alice}\ \mathbf{speaksfor}\ \mathsf{BankA.Alice}\ \mathbf{true}) \vdash \mathsf{BankA}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{signed}$$

$$\dfrac{\dfrac{\dfrac{}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathbf{action}(\ldots)\ \mathbf{true}}\ \textbf{hyp}}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathsf{BankA}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{aff}}{\mathcal{C}_{0-4};\mathsf{BankA}\ \mathbf{says\ action}(\ldots)\ \mathbf{true} \vdash \mathsf{BankA}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{saysL}$$

$$\dfrac{\cdots}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_1 \vdash \mathsf{BankA}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{delegate}$$

$\mathcal{D}_2$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_3}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_1 \vdash \mathsf{BankA}\ \mathbf{affirms\ action}(\ldots)}}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_1 \vdash \mathsf{BankA}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{saysR}}{\mathcal{C}_{0-4};\mathcal{C}_5 \vdash \mathsf{BankA}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{copy}'$$

$$\dfrac{\dfrac{\dfrac{}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathbf{action}(\ldots)\ \mathbf{true}}\ \textbf{hyp}}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathsf{ACH.BC.BankA}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{aff}}{\mathcal{C}_{0-4};\mathsf{ACH.BC.BankA}\ \mathbf{says\ action}(\ldots)\ \mathbf{true} \vdash \mathsf{ACH.BC.BankA}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{saysL}$$

$$\dfrac{\cdots}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{A}_2 \vdash \mathsf{ACH.BC.BankA}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{speaksfor}$$

$\mathcal{D}_1$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_2}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{A}_2 \vdash \mathsf{ACH.BC.BankA}\ \mathbf{affirms\ action}(\ldots)}}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{A}_2 \vdash \mathsf{ACH.BC.BankA}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{saysR}}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{A}_2, \mathbf{delegate}\ (\mathsf{ACH.BC}, \mathsf{ACH.BC.BankA}, pay)\ \mathbf{true} \vdash \mathsf{ACH.BC}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{signed}}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_4,\mathcal{A}_2 \equiv (\mathsf{BankA}\ \mathbf{speaksfor}\ \mathsf{ACH.BC.BankA}\ \mathbf{true}) \vdash \mathsf{ACH.BC}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{signed}$$

$$\dfrac{\dfrac{\dfrac{}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathbf{action}(\ldots)\ \mathbf{true}}\ \textbf{hyp}}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathsf{ACH.BC}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{aff}}{\mathcal{C}_{0-4};\mathsf{ACH.BC}\ \mathbf{says\ action}(\ldots)\ \mathbf{true} \vdash \mathsf{ACH.BC}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{saysL}$$

$$\dfrac{\cdots}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_4,\mathcal{C}_2 \vdash \mathsf{ACH.BC}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{delegate}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_4,\mathcal{C}_2 \vdash \mathsf{ACH.BC}\ \mathbf{affirms\ action}(\ldots)}}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_4,\mathcal{C}_2 \vdash \mathsf{ACH.BC}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{saysR}}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_4 \vdash \mathsf{ACH.BC}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{copy}'}{\mathcal{C}_{0-4};\mathcal{C}_5 \vdash \mathsf{ACH.BC}\ \mathbf{says\ action}(\ldots)\ \mathbf{true}}\ \textbf{copy}'$$

$$\dfrac{\dfrac{\dfrac{}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathbf{action}(\ldots)\ \mathbf{true}}\ \textbf{hyp}}{\mathcal{C}_{0-4};\mathbf{action}(\ldots)\ \mathbf{true} \vdash \mathsf{ACH}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{aff}}{\mathcal{C}_{0-4};\mathsf{ACH}\ \mathbf{says\ action}(\ldots)\ \mathbf{true} \vdash \mathsf{ACH}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{saysL}$$

$$\dfrac{\cdots}{\mathcal{C}_{0-4};\mathcal{C}_5, \mathbf{delegate}\ (\mathsf{ACH}, \mathsf{ACH.BC}, pay)\ \mathbf{true} \vdash \mathsf{ACH}\ \mathbf{affirms\ action}(\ldots)}\ \textbf{delegate}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_3 \vdash \mathsf{ACH}\ \mathbf{affirms\ action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)}\ \textbf{signed}}{\mathcal{C}_{0-4};\mathcal{C}_5,\mathcal{C}_3 \vdash \mathsf{ACH}\ \mathbf{says\ action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)\ \mathbf{true}}\ \textbf{saysR}}{\mathcal{C}_{0-4};\mathcal{C}_5 \vdash \mathsf{ACH}\ \mathbf{says\ action}(pay, \langle \mathsf{Bob}, \$100\rangle, nonce)\ \mathbf{true}}\ \textbf{copy}'$$

Figure 5. Proof of ACH says action(...) true

$$
\begin{aligned}
\mathcal{C}_0 &= \mathsf{Calendar}\ \mathbf{signed}_{\mathsf{RCal}}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Monday}, \mathit{0800–0900}\rangle)\\
\mathcal{C}_1 &= \mathsf{Calendar}\ \mathbf{signed}_{\mathsf{RCal}}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Wednesday}, \mathit{0800–0900}\rangle)\\
\mathcal{C}_2 &= \mathsf{Calendar}\ \mathbf{signed}_{\mathsf{RCal}}\ \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Friday}, \mathit{0800–0900}\rangle)\\
\mathcal{C}_3 &= \mathsf{Registrar}\ \mathbf{signed}_{\mathsf{RSeat}}\ \mathbf{action}(\mathit{seat}, \langle F'05, \mathit{CS101}\rangle, \mathit{nonce})\\
\mathcal{C}_4 &= \mathsf{Registrar}\ \mathbf{signed}_{\mathsf{RCredit}}\mathbf{delegate}\ (\mathsf{Registrar}, \mathsf{Alice}, \mathit{credit\_hours})\\
\mathcal{C}_5 &= \mathsf{Alice}\ \mathbf{signed}\ \mathbf{action}(\mathit{credit\_hours}, \langle \mathsf{Alice}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce})\\
\mathcal{C}_6 &= \mathsf{Registrar}\ \mathbf{signed}\ (\forall A.(\mathsf{Calendar}\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle A, F'05, \mathit{Monday}, \mathit{0800–0900}\rangle)\\
&\qquad \wedge\ \mathsf{Calendar}\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle A, F'05, \mathit{Wednesday}, \mathit{0800–0900}\rangle)\\
&\qquad \wedge\ \mathsf{Calendar}\ \mathbf{says}\ \mathbf{action}(\mathit{timeslot}, \langle A, F'05, \mathit{Friday}, \mathit{0800–0900}\rangle)\\
&\qquad \wedge\ \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}(\mathit{seat}, \langle F'05, \mathit{CS101}\rangle, \mathit{nonce})\\
&\qquad \wedge\ \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}(\mathit{credit\_hours}, \langle A, F'05, \mathit{4\ credits}\rangle, \mathit{nonce}))\\
&\qquad \rightarrow \mathbf{action}(\mathit{register}, \langle A, \mathit{CS101}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce}))\\
\mathcal{C}_7 &= \mathsf{RCal}\ \mathbf{signed}\ \langle \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Monday}, \mathit{0800–0900}\rangle), \mathcal{M},\\
&\quad \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce})\rangle\\
\mathcal{C}_8 &= \mathsf{RCal}\ \mathbf{signed}\ \langle \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Wednesday}, \mathit{0800–0900}\rangle), \mathcal{M},\\
&\quad \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce})\rangle\\
\mathcal{C}_9 &= \mathsf{RCal}\ \mathbf{signed}\ \langle \mathbf{action}(\mathit{timeslot}, \langle \mathsf{Alice}, F'05, \mathit{Friday}, \mathit{0800–0900}\rangle), \mathcal{M},\\
&\quad \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce})\rangle\\
\mathcal{C}_{10} &= \mathsf{RSeat}\ \mathbf{signed}\ \langle \mathbf{action}(\mathit{seat}, \langle F'05, \mathit{CS101}\rangle, \mathit{nonce}), \mathcal{M},\\
&\quad \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce})\rangle\\
\mathcal{C}_{11} &= \mathsf{RCredit}\ \mathbf{signed}\ \langle \mathbf{delegate}\ (\mathsf{Registrar}, \mathsf{Alice}, \mathit{credit\_hours}), \mathcal{M},\\
&\quad \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}(\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce})\rangle
\end{aligned}
$$

Figure 6. Credentials that allow Alice to register for a class

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.

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 to prove:

$$
\begin{aligned}
\mathcal{G} = {} & \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}\\
& (\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce})\ \mathbf{true}
\end{aligned}
$$

When Alice's registration period began, she was given credentials $\mathcal{C}_0$–$\mathcal{C}_2$, (see Figure 6) 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 that Alice must prove before registration.

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 $\mathbf{action}()$ statements where they are unnecessary (e.g., $\mathcal{C}_0$–$\mathcal{C}_2$).

Alice now has enough credentials to prove

$$
\begin{aligned}
\mathcal{M} : {} & \mathsf{Registrar}\ \mathbf{says}\ \mathbf{action}\\
& (\mathit{register}, \langle \mathsf{Alice}, \mathit{CS101}, F'05, \mathit{4\ credits}\rangle, \mathit{nonce})\ \mathbf{true}
\end{aligned}
$$

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.

The full proof is similar to those done in Appendices B.1 and B.2, though much larger. We do not show the full proof here. Upon checking the proof, the registrar would then register Alice for CS101, as desired.