An Introduction to Proof-Carrying Authorization

Deepak Garg

December 11, 2007

1 Introduction

In the interest of security, most computer systems restrict operations on resources like files and memory locations to specific users. This is called access control. For example, a file on most systems can be accessed by the root user, owner of the file and anyone specifically allowed by the owner only. Similarly, only the root user and owning process can write to a memory location. More interesting cases arise in distributed settings, as for instance when an individual accesses her bank statements online. In this case the bank's website must determine whether the requesting client is authorized to view the statements or not.

Irrespective of the exact setting, access control is usually engineered as follows: each operation on the controlled resource is intercepted by a controlling program (called *reference monitor*) that allows the operation to succeed only if the calling program (called *requester*) has sufficient permissions to complete it. In the case of file system and memory, the reference monitor is the operating system; for the online banking system, the web-server hosting the site acts as the reference monitor.

In order to make a decision, the reference monitor must accomplish two smaller tasks: *authentication* and *authorization*. Authentication is the problem of determining the identity of the requesting principal, while authorization is the problem of deciding whether this principal should be allowed access. The latter decision is based on a set of rules called the *policy*.

For centralized problems like memory access or file access in local systems such as ext2, authentication is relatively straightforward: the operating system already knows the identity of the requesting process (and related details like process owner, process group, etc.). Authentication is much harder in distributed scenarios. For such settings several methods are used. These include passwords, challenge-response protocols, and IP address determination. A highly secure website will usually use a combination of these methods. A bank's website may, for instance, request a login name and password, verify the user's IP address, and use a challenge-response protocol to establish a session. Given such a variety of possible solutions, authentication is not a very difficult task and can be realized efficiently in practice.

Authorization, or the problem of deciding whether the policy allows access to the requesting principal, is more complex. This complexity arises due to two main reasons: (1) it is difficult (except in the simplest cases) to specify the policy in a machine parsable, unambiguous manner; in particular, textual descriptions do not suffice and (2) owing to the complexity of most policies, it is hard to automatically decide whether access is allowed, even if the policy has an unambiguous specification. Solving the first problem requires a *language* in which policies can be written, while the second demands an inference or reasoning system over the language. The language must be expressive enough for encoding policies of the application at hand, and both the language and inference system must be unambiguous and machine tractable. A number of proposals addressing one or both these issues have been made. Particularly relevant to our discussion are those based on formal logic [1–4, 8–10, 12] since logic provides a rigorous machine parsable syntax (language) and a deductive system to reason about it.

2 Proof-carrying Authorization

Even after policies have been unambiguously written in a logic (or some other language) and the inference mechanism completely defined, there still remains the question of actually *programming* the reference monitor to use inference on the given policy to make decisions. As it turns out, this is quite difficult in the general case: even for the simplest logics (e.g., classical propositional logic, which is too simple to be of use here), the problem of deciding whether a particular access should be allowed from a given policy is NP-complete. For the logic that we consider (first-order indexed lax logic), the problem is undecidable, and at least PSPACE-complete on a smaller useful fragment. As a result, programming a generic reference monitor that can make access decisions is extremely difficult and inefficient.

Let us examine this complexity in a little more detail. We take the example of the file system, where a file can be accessed by a principal K due to any of three possible reasons: (1) K is the root user, (2) K owns the file, or (3) the owner explicitly allowed access to K. Suppose a principal Alice tries to read the file. In order to decide if she is allowed access, the reference monitor *might have to explore all three possibilities*. It must first check if Alice is the root user, then if that fails, it must check if Alice owns the file, and if that fails too, it must check if the owner has allowed Alice access. This kind of exploration based reasoning makes it very difficult (in terms of time and feasibility) for the reference monitor to decide if access is allowed or not. In some cases, there may not even be a clear search strategy. Architectures that require the access control monitor to reason whether access is allowed are therefore either impractical or restricted to very simple policies.

In sharp contrast, the principal making the request generally knows why it should be allowed access. In the above example, for instance, Alice would usually know that she is either the owner or root user, or that she has been

explicitly given access by the owner. As a result it is *much easier for Alice to reason why she has access.* In particular, she does not have to go through the exploratory search that the reference monitor must. This situation is quite general. In most cases, it is computationally straightforward (often trivial) for the requesting principal to know why it should have access, but the reference monitor must generically reason (with an exponential or incomplete algorithm) to reach the same conclusion.

In the interest of reducing computation, then, there is quintessential need for a mechanism by which the requester can *convince* the reference monitor why it should be given access. If such a mechanism were provided, the requester could determine why it had access and convince the reference monitor of the same. This would save a lot of computation on part of the reference monitor, increasing the throughput of the system. Fortunately, logic provides such a convenient mechanism in the form of *proofs.* A proof is a textual and completely rigorous representation of a logical inference. The requester provides the reference monitor a proof that it is allowed access. The monitor only has to verify that the proof is correct. This does not require any exploration, and is computationally straightforward (linear in the size of the proof). This idea has been implemented in several systems, most recently in the Grey system at CMU [5, 6], and is called proof-carrying authorization (PCA) [3, 5, 7].

In summary, there is an irony in reasoning with policies. The requester can reason easily why it has access, but the reference monitor cannot, yet it is the latter which must be convinced that this is the case. If policies are written in a logic, the requester can construct a proof witnessing the fact that the policies imply access for it, and submit it with the access request. The reference monitor verifies that the proof is correct, and if so, allows access, else denies it. This approach, called proof-carrying authorization, is superior to those in which the reference monitor reasons with the policy directly, because generic inference is computationally prohibitive, whereas both proof-construction (at the requester's end) and proof-verification (at the reference monitor's end) can usually be implemented very efficiently.

It is this proof-carrying authorization architecture that we illustrate in these notes. We describe one logic [8] in which policies for access control can be written and reasoned with. We also describe the structure of proofs and how they can be verified by the reference monitor. Throughout the notes, we use the following example to illustrate various concepts. We completely ignore issues of authentication, assuming that they are resolved through some external mechanism.

Example 1. The ACM digital library (represented by the principal ACM) is an online repository of computer science articles. All university students are allowed free download access. However, ACM cannot determine who a student is, and who is not. To this end, it allows any university to buy a group subscription, after which the university can tell ACM who its students are, thus allowing the latter free downloads. We assume that university CMU has a subscription. A textual description of the relevant policies of ACM is the following:

1. [ACM says] If principal X is a student, then X can download any article.
2. [ACM says] If CMU says that principal X is a student, then I believe that X is a student.

We explicitly write [ACM says] to indicate that these are the policies of the ACM digital library. Such a qualification is necessary because in a distributed setting there may be many principals with different policies, and they may not always agree. For example, some principal Bob may believe that he is allowed to download, but ACM may not.

We assume that principal Alice is a CMU student. Thus we have one more policy:

3. [CMU says] Alice is a student.

The ACM library's website works on a proof-carrying authorization architecture. Along with a request, the requesting principal X must submit a proof that it is allowed to download, i.e., a proof that "[ACM believes] X is allowed to download" is derivable from the policy. Observe that the requester X must provide enough evidence to convince ACM that it is allowed to download (hence the annotation [ACM believes]). There is no obligation on part of the requester to establish to any other principal that this is the case.

3 Syntax of the Logic

The logic we consider for PCA is an extension of first-order intuitionistic logic. This is only one possible logic that may be used for the purpose; several others have been proposed and used in the past. We omit a discussion of relative merits of these logics, partly in the interest of not digressing from the topic at hand, and partly because there is no standard way to compare the logics, except by actual empirical evidence.

The syntax of a logic can be divided into two main categories: formulas and proof-terms. Formulas are analogous to statements in English, and form the language in which policies are written. Proof-terms are a syntactic representation of reasoning performed, and they are analogous to mathematical proofs that we write. In addition, we have principals which form a third syntactic category. We use the following notation: A, B, C for formulas, M, N, E for proof-terms, and K, X and concrete names Alice, Bob, Charlie, ... for principals. We also use lowercase letters x, y as variables that range over principals.[1] We describe the formulas first, leaving a discussion of proof-terms to the next section.

The most basic formulas are atomic formulas (also called predicates), which express properties of principals, or describe relations between them. Referring to our earlier example, we could write the fact that "X is allowed to download" as the formula `canDownload`(X). Similarly, the fact that X is a student

[1] The logic allows variables (and quantification) over all kinds of entities, not just principals. Since we have no occasion to use entities other than principals, we restrict ourselves to this limited formulation.

can be written $\texttt{isStudent}(X)$. In general an atomic formula has the form $\texttt{pred}(t_1, \ldots, t_n)$, where $\texttt{pred}$ is the name of the predicate and $t_1, \ldots, t_n$ are objects that are related by the predicate.

Complex formulas can be constructed by combining simpler ones using *connectives*. There are three connectives we use. If A and B are any formulas, then the formula $A \supset B$ (read A implies B) means that whenever A holds then B holds. It is analogous to "if - then" constructions in spoken language (*if A holds then B holds*).

If A is a formula, then $\forall x.A$ (read for all x, A) is a formula meaning that for "for every principal x, it is the case that A holds". For instance, the formula $\forall x.\texttt{isStudent}(x)$ means that every principal is a student. Similarly, $\forall x.(\texttt{isStudent}(x) \supset \texttt{canDownload}(x))$ means that for every individual x, it is the case that if x is a student, then x can download. Or, in simpler words, every student is allowed to download.

We need one more connective to represent qualifications like [ACM says] and [ACM believes]. If K is a principal, and A is a formula, then we write K $\texttt{says}$ A to mean both that K states that formula A holds and that K believes that formula A holds.[2] For example, the formula ACM $\texttt{says}$ $\forall x.(\texttt{isStudent}(x) \supset \texttt{canDownload}(x))$ represents ACM's first policy.

The following describes the syntax of formulas, written as a BNF grammar. We use P to denote arbitrary atomic formulas.

$$A, B \ ::= \ P \mid A \supset B \mid \forall x.A \mid K \ \texttt{says} \ A$$

Example 2. The three policies of ACM and CMU from the example 1 written as logical formulas are the following:

$$\begin{array}{lcl} p1 & : & \text{ACM } \texttt{says } \forall x.\ (\texttt{isStudent}(x) \supset \texttt{canDownload}(x)) \\ p2 & : & \text{ACM } \texttt{says } \forall x.\ ((\text{CMU } \texttt{says } \texttt{isStudent}(x)) \supset \texttt{isStudent}(x)) \\ p3 & : & \text{CMU } \texttt{says } \texttt{isStudent}(\text{Alice}) \end{array}$$

We give the policies names ($p1$, $p2$ and $p3$), so that they can be referred to in a proof. In order to download a file, a principal Alice must submit a proof that the policies logically entail the following formula:

$$\text{ACM } \texttt{says } \texttt{canDownload}(\text{Alice})$$

This formula states that ACM believes that Alice is allowed to download.

4 Inference, Proofs and Proof-terms

Reasoning in a logic is very rigorous, subject to well-defined rules, called *inference rules*. The question is: what do these rules apply on, what is their form,

[2] It is possible to conceive logics in which statements and beliefs are distinct, but we do not have occasion to use such distinctions. Hence in our logic we identify the statements and beliefs of each principal.

and how do we use them? Further, what is the formal definition of a proof, and what is a proof-term? In this section we answer these questions.

We start with an informal description, using our digital library's example. As stated in example 2, Alice needs to prove that ACM says canDownload(Alice). Since the entity we prove is a formula, our first instinct might be to say that inference rules should apply over formulas, and should let us conclude formulas from other formulas. For instance, in the example at hand, we could start by assuming the formulas named $p1$, $p2$ and $p3$ and apply the rules to finally conclude ACM says canDownload(Alice). While not entirely incorrect, this intuition of specifying rules that allow us to conclude formulas from other (already established or assumed) formulas is incomplete. Using this approach, we can capture the meaning of most parts of the logic, but not all.

In order to capture the entire logic, we introduce the notion of *judgments* or *sequents* (denoted by the letter J). Judgments are composed of formulas, and rules are defined over judgments. Each rule has the following form: *if judgments $J_1, \ldots, J_n$ can be established, then judgment J can also be established.* This is commonly written as follows, where (name) is a name given to the rule:

$$\frac{J_1 \ldots J_n}{J}\text{name}$$

The judgments $J_1, \ldots, J_n$ written above the line are called the premises of the rule, and the judgment J written below the line is called the conclusion of the rule. The meaning of the rule is the following: if judgments $J_1, \ldots, J_n$ have been established, then J can also be established. Thus reasoning in the logic is about judgments, not formulas.[3] The number of premises n in a rule may be zero, in which case the conclusion of the rule is self-evident. Such rules (and, perhaps confusingly, their corresponding conclusions) are called *axioms*.

We say that a judgment J is *derivable* in the logic, or that J is *provable* in the logic, if starting from axioms, we can apply a finite number of inference rules to derive more and more judgments, ending in the judgment J. The specific sequence of steps used is called a *proof*. A proof can be written succinctly as a figure, as the following example illustrates.

Example 3. Suppose that in some logic, we have five judgments J_1, J_2, J_3, J_4, J_5, and the following inference rules:

$$\frac{}{J_1}\text{Ax1} \qquad \frac{}{J_2}\text{Ax2} \qquad \frac{J_1 \qquad J_2}{J_3}\text{R1} \qquad \frac{J_2}{J_4}\text{R2} \qquad \frac{J_1 \qquad J_3}{J_4}\text{R3}$$

The rules Ax1 and Ax2 are axioms. Correspondingly, the concluding judgments J_1 and J_2 are self-evident. The following is a proof that the judgment J_4 is provable (or derivable) in the logic:

$$\dfrac{\dfrac{}{J_2}\ \text{Ax2}}{J_4}\ \text{R2}$$

[3]This distinction is due to Martin Löf [11]. Prior to this work, judgments and formulas were identified.

Observe what this proof says: first use the rule (axiom) Ax2 to conclude J_2. Then use the rule R2 to conclude J_4. Here is *another* proof that J_4 is provable:

$$\dfrac{\dfrac{}{J_1}\ \text{Ax1} \qquad \dfrac{\dfrac{}{J_1}\ \text{Ax1} \qquad \dfrac{}{J_2}\ \text{Ax2}}{J_3}\ \text{R1}}{J_4}\ \text{R3}$$

This proof states the following: First use Ax1 to conclude J_1. Also, use Ax2 to conclude J_2. Now combine both of these using rule R1 to conclude J_3. Then use rule R3 to conclude J_4. Observe that the axiom Ax1 is used twice in this proof.

As this example illustrates, a single judgment (J_4 here) may have several proofs. For the purposes of proof-carrying authorization, any single proof suffices, since we are interested only in the provability of a judgments, not their specific proofs.

Exercise. For the above example, show that the judgment J_3 has only one proof, and write it down. Also explain why the judgment J_5 is unprovable.

4.1 Judgments

The above discussion describes judgments, rules and inference in general. Each logic requires specific judgments and specific inference rules. For our logic, we need two forms of judgments:

$$\mathcal{P} \Longrightarrow M : A$$
$$\mathcal{P} \Longrightarrow E : K\ affirms\ A$$

$\mathcal{P}$ denotes a set of formulas together with names for them (like $p1$, $p2$, $p3$). These are meant to represent the policy. Technically they are called *assumptions* or *hypotheses*. In the example of the digital library, we could choose:

$$\begin{array}{rcrcl} \mathcal{P}_0 & = & p1 & : & \mathsf{ACM}\ \mathtt{says}\ \forall x.\ (\mathtt{isStudent}(x) \supset \mathtt{canDownload}(x)), \\ & & p2 & : & \mathsf{ACM}\ \mathtt{says}\ \forall x.\ ((\mathsf{CMU}\ \mathtt{says}\ \mathtt{isStudent}(x)) \supset \mathtt{isStudent}(x)), \\ & & p3 & : & \mathsf{CMU}\ \mathtt{says}\ \mathtt{isStudent}(\mathsf{Alice}) \end{array}$$

K and A stand for principals and formulas, as described in section 3. The symbol $\Longrightarrow$ is called the sequent arrow. M and E are proof-terms, which are succinct representations of proofs. These two forms of judgments are templates for creating judgments; the latter may be obtained by substituting specific instances of $\mathcal{P}$, M, E, K and A.

As we describe these judgments further, an essential point to keep in mind is the following:

Fact 1. *If a judgment is provable in our logic, it has exactly one proof.* Further this proof can be reconstructed from the proof-term embedded in the judgment (M or E). Thus the proof and proof-term can be considered synonymous

(The fact that the proof-term represents the proof completely is explained in section 5.)

The intuitive explanation of the first judgment form is the following: if provable, $\mathcal{P} \Longrightarrow M : A$ states that "from the policy $\mathcal{P}$, the formula A logically follows. Further, M is a complete representation of a proof that this is the case."

The second judgment is similarly explained: if provable, $\mathcal{P} \Longrightarrow E : K\ affirms\ A$ means that "from the policy $\mathcal{P}$, it follows that K must believe formula A. Further, E is a complete representation of a proof that this is the case."

Example 4. We revise example 2 of the ACM digital library. In order to download from the website, Alice must demonstrate that the following sequent is provable for some proof-term M ($\cdot$ represents the empty set):

$$\cdot \Longrightarrow M : \mathsf{ACM}\ \mathtt{says}\ \mathtt{canDownload}(\mathsf{Alice})$$

The specific proof-term does not matter. Intuitively, we are saying that Alice must show that there is *at least* one reason (proof-term) that convinces ACM that she is allowed to download. Any reason will work, so long as it is correct. Of course, if we fix a reason M, then by fact 1, there is at most one formal proof that allows us to conclude this judgment.

As it turns out, the above judgment is actually unprovable. The correct strategy for Alice is to first show that there is a proof-term M' such that the following judgment is provable ($\mathcal{P}_0$ is set of relevant policies described earlier):

$$\mathcal{P}_0 \Longrightarrow M' : \mathsf{ACM}\ \mathtt{says}\ \mathtt{canDownload}(\mathsf{Alice})$$

If she can do this, then ACM will believe that Alice can download, provided the policies in $\mathcal{P}_0$ hold. In order to convince ACM of the latter, Alice must submit further evidence. This is discussed in section 5.

Proof-terms. The exact syntax of proof-terms is described below. The symbols let, in,$\langle K \rangle$, λ, Λ, and aff are keywords, occurring in proof-terms only. As may be expected, proof-terms M and E are allowed to mention names for assumptions (such as $p1$, $p2$, and $p3$) that occur in $\mathcal{P}$. We use the letter p to refer to these names generically.

$$\begin{array}{lll} M, N & ::= & p \mid \lambda p : A.M \mid M\ N \mid \Lambda x.M \mid M\ K \mid \langle K \rangle E \\ E & ::= & aff\langle K \rangle\ M \mid let\langle K \rangle\ p = M\ in\ E \end{array}$$

4.2 Inference Rules

We now describe the inference rules of our logic. Just like the judgments, rules are written in a generic form which can be instantiated by substituting specific values for meta-variables like $\mathcal{P}$, A, etc. As a whole, there are eight forms of inference rules, of which we describe six here. These six are the ones that

are commonly used for reasoning from policies. The remaining two are used in cases that occur very rarely in practice. As we describe the six forms of rules, we also construct a proof-term M' such that $\mathcal{P}_0 \Longrightarrow M' : \mathsf{ACM}\ \mathtt{says}\ \mathtt{canDownload}(\mathsf{Alice})$ is derivable in the logic. We start with the only axiom, which we call USE.

$$\frac{}{\mathcal{P},\ p : A \Longrightarrow p : A}\ \text{USE}$$

This axiom rule states the following: if the hypotheses contain the formula A (i.e., A is already assumed in the policy), then A follows from the policy. Further, if p is the name of the policy A, then the proof-term representing this (trivial) proof is also called p.

For our example, we can conclude the following judgment by direct application of the USE rule.

$$\mathcal{P}_0 \Longrightarrow p1 : \mathsf{ACM}\ \mathtt{says}\ \forall x.\ (\mathtt{isStudent}(x) \supset \mathtt{canDownload}(x)) \tag{1}$$

Similarly, if we define

$$\begin{array}{rcl} \mathcal{P}_1 & = & \mathcal{P}_0,\ p4 : \forall x.\ (\mathtt{isStudent}(x) \supset \mathtt{canDownload}(x)) \\ \mathcal{P}_2 & = & \mathcal{P}_1,\ p5 : \forall x.\ ((\mathsf{CMU}\ \mathtt{says}\ \mathtt{isStudent}(x)) \supset \mathtt{isStudent}(x)) \end{array}$$

then we can also use the USE rule to conclude that

$$\mathcal{P}_1 \Longrightarrow p2 : \mathsf{ACM}\ \mathtt{says}\ \forall x.\ ((\mathsf{CMU}\ \mathtt{says}\ \mathtt{isStudent}(x)) \supset \mathtt{isStudent}(x)) \tag{2}$$

$$\mathcal{P}_2 \Longrightarrow p3 : \mathsf{CMU}\ \mathtt{says}\ \mathtt{isStudent}(\mathsf{Alice}) \tag{3}$$

$$\mathcal{P}_2 \Longrightarrow p4 : \forall x.\ (\mathtt{isStudent}(x) \supset \mathtt{canDownload}(x)) \tag{4}$$

$$\mathcal{P}_2 \Longrightarrow p5 : \forall x.\ ((\mathsf{CMU}\ \mathtt{says}\ \mathtt{isStudent}(x)) \supset \mathtt{isStudent}(x)) \tag{5}$$

The next rule deals with the connective $\forall x.A$. It states that if we can infer that $\forall x.A$ follows logically from hypotheses $\mathcal{P}$ (with proof term M), then from the same hypotheses, $A[K/x]$ must also follow for any principal K. We write the proof-term representing the new proof as $M\ K$. ($A[K/x]$ is notation for substituting K for the variable x in A.) This captures the meaning of the universal quantifier. The name of the rule $\forall E$ is suggestive of what it does: it *eliminates* the quantifier from the formula $\forall x.A$ (E stands for eliminate).

$$\frac{\mathcal{P} \Longrightarrow M : \forall x.A}{\mathcal{P} \Longrightarrow M\ K : A[K/x]}\ \forall E$$

As specific instances, we can derive the following two judgments by applying the rule $\forall E$ to the judgments (4) and (5), substituting Alice for x in each case.

$$\mathcal{P}_2 \Longrightarrow p4\ \mathsf{Alice} : (\mathtt{isStudent}(\mathsf{Alice}) \supset \mathtt{canDownload}(\mathsf{Alice})) \tag{6}$$

$$\mathcal{P}_2 \Longrightarrow p5\ \mathsf{Alice} : ((\mathsf{CMU}\ \mathtt{says}\ \mathtt{isStudent}(\mathsf{Alice})) \supset \mathtt{isStudent}(\mathsf{Alice})) \tag{7}$$

Our next rule deals with the connective $A \supset B$. It states that if we can infer that from some set of hypotheses $\mathcal{P}$, both $A \supset B$ and A follow logically, then we can also infer that from hypotheses $\mathcal{P}$, B follows logically. Further, if the proof-terms representing the proofs of the two premises are M and N, then we write the proof-term representing the new proof as $M\ N$.

$$\frac{\mathcal{P} \Longrightarrow M : A \supset B \qquad \mathcal{P} \Longrightarrow N : A}{\mathcal{P} \Longrightarrow M\ N : B} \supset E$$

For example, we can apply the rule $\supset E$ to the judgments (7) and (3) to derive the following judgment:

$$\mathcal{P}_2 \Longrightarrow (p5\ \mathsf{Alice})\ p3 : \texttt{isStudent}(\mathsf{Alice}) \tag{8}$$

Similarly, we can apply $\supset E$ to the judgments (6) and (8) to derive:

$$\mathcal{P}_2 \Longrightarrow (p4\ \mathsf{Alice})\ ((p5\ \mathsf{Alice})\ p3) : \texttt{canDownload}(\mathsf{Alice}) \tag{9}$$

The following rule relates the two forms of judgments. It states that if we can infer that A follows logically from some hypotheses $\mathcal{P}$ then for any principal K, $K\ affirms\ A$ also follows from the same hypotheses. If the proof-term representing the proof of the premise is M, then that representing the proof of the conclusion is written $aff\langle K\rangle\ M$.

$$\frac{\mathcal{P} \Longrightarrow M : A}{\mathcal{P} \Longrightarrow aff\langle K\rangle\ M : K\ affirms\ A}\ \text{AFF}$$

For example, we can apply this rule to judgment (9), with $K = \mathsf{ACM}$ to conclude that:

$$\mathcal{P}_2 \Longrightarrow aff\langle \mathsf{ACM}\rangle\ (p4\ \mathsf{Alice})\ ((p5\ \mathsf{Alice})\ p3) : \mathsf{ACM}\ affirms\ \texttt{canDownload}(\mathsf{Alice}) \tag{10}$$

Now we describe the rules related to the connective K `says` A. The following rule, which is perhaps the most non-intuitive rule, states that if from some hypotheses $\mathcal{P}$, K `says` A follows logically, and by assuming A in addition to $\mathcal{P}$, $K\ affirms\ C$ follows logically, then $K\ affirms\ C$ follows from the hypotheses $\mathcal{P}$ (without the assumption A). Further, if the proof-term corresponding to the two premises are M and E respectively, and A is named p in E, then the proof-term representing the proof of the conclusion is written $let\langle K\rangle\ p = M\ in\ E$.

$$\frac{\mathcal{P} \Longrightarrow M : K\ \texttt{says}\ A \qquad \mathcal{P},\ p : A \Longrightarrow E : K\ affirms\ C}{\mathcal{P} \Longrightarrow let\langle K\rangle\ p = M\ in\ E : K\ affirms\ C}\ \text{SAYS-E}$$

As an instance, we can apply rule SAYS-E to the judgments (2) and (10) to conclude that

$$\begin{aligned}\mathcal{P}_1 \Longrightarrow\ & let\langle \mathsf{ACM}\rangle\ p5 = p2\ in \\ & aff\langle \mathsf{ACM}\rangle\ (p4\ \mathsf{Alice})\ ((p5\ \mathsf{Alice})\ p3) : \mathsf{ACM}\ affirms\ \texttt{canDownload}(\mathsf{Alice})\end{aligned} \tag{11}$$

We can apply the rule SAYS-E again to judgments (1) and (11) to conclude the following judgment.

$$\begin{aligned}\mathcal{P}_0 \Longrightarrow\ & let\langle \mathsf{ACM}\rangle\ p4 = p1\ in \\ & let\langle \mathsf{ACM}\rangle\ p5 = p2\ in \\ & aff\langle \mathsf{ACM}\rangle\ (p4\ \mathsf{Alice})\ ((p5\ \mathsf{Alice})\ p3) : \mathsf{ACM}\ affirms\ \mathtt{canDownload}(\mathsf{Alice})\end{aligned} \tag{12}$$

Our last rule, SAYS-I is shown below. It states the following: if from hypotheses $\mathcal{P}$, it follows logically that $K\ affirms\ A$, then from the same hypotheses, K says A also follows. If the proof-term representing the proof of the premise of the rule is E, then that representing the conclusion is $\langle K\rangle\ E$.

$$\frac{\mathcal{P} \Longrightarrow E : K\ affirms\ A}{\mathcal{P} \Longrightarrow \langle K\rangle\ E : K\ \mathtt{says}\ A}\ \text{SAYS-I}$$

For example, we can apply the rule SAYS-I to judgment (12) to conclude that:

$$\begin{aligned}\mathcal{P}_0 \Longrightarrow\ & \langle \mathsf{ACM}\rangle \\ & let\langle \mathsf{ACM}\rangle\ p4 = p1\ in \\ & let\langle \mathsf{ACM}\rangle\ p5 = p2\ in \\ & aff\langle \mathsf{ACM}\rangle\ (p4\ \mathsf{Alice})\ ((p5\ \mathsf{Alice})\ p3) : \mathsf{ACM}\ \mathtt{says}\ \mathtt{canDownload}(\mathsf{Alice})\end{aligned} \tag{13}$$

The reader may observe that judgment (13) is what Alice wanted to establish (see example 4). Thus in that example, Alice can successfully download the article she wants from ACM's digital library, if she can establish that all policies mentioned in $\mathcal{P}_0$ are valid.

This concludes our description of inference rules of the logic, with the exception of two rules which are rarely used in practice. Figure 1 summarizes all the rules, including the two not described here ($\forall I$ and $\supset I$).

5 Verification: From Proof-terms to Proofs

Having described the inference rules, proofs and proof-terms in the logic, the natural next step is to describe the manner in which they are used. In the context of proof-carrying authorization, the exact question is: what should Alice submit to ACM's website before she is allowed to download? As we have seen in example 4, she needs to demonstrate to the website that there a proof-term M such that the following judgment is provable:

$$\cdot \Longrightarrow M : \mathsf{ACM}\ \mathtt{says}\ \mathtt{canDownload}(\mathsf{Alice}) \tag{14}$$

We also saw in the last section that there a proof-term M' such that the following judgment is provable:

$$\mathcal{P}_0 \Longrightarrow M' : \mathsf{ACM}\ \mathtt{says}\ \mathtt{canDownload}(\mathsf{Alice}) \tag{15}$$

$$\frac{}{\mathcal{P},\ p : A \Longrightarrow p : A}\ \text{USE} \qquad \frac{\mathcal{P} \Longrightarrow M : A}{\mathcal{P} \Longrightarrow aff\langle K\rangle\ M : K\ affirms\ A}\ \text{AFF}$$

$$\frac{\mathcal{P} \Longrightarrow M : A}{\mathcal{P} \Longrightarrow \Lambda x.M : \forall x.A}\ \forall I^* \qquad \frac{\mathcal{P} \Longrightarrow M : \forall x.A}{\mathcal{P} \Longrightarrow M\ K : A[K/x]}\ \forall E$$

$$\frac{\mathcal{P},\ p : A \Longrightarrow M : B}{\mathcal{P} \Longrightarrow \lambda p : A.M : A \supset B}\ \supset I \qquad \frac{\mathcal{P} \Longrightarrow M : A \supset B \qquad \mathcal{P} \Longrightarrow N : A}{\mathcal{P} \Longrightarrow M\ N : B}\ \supset E$$

$$\frac{\mathcal{P} \Longrightarrow E : K\ affirms\ A}{\mathcal{P} \Longrightarrow \langle K\rangle\ E : K\ \mathtt{says}\ A}\ \text{SAYS-I}$$

$$\frac{\mathcal{P} \Longrightarrow M : K\ \mathtt{says}\ A \qquad \mathcal{P},\ p : A \Longrightarrow E : K\ affirms\ C}{\mathcal{P} \Longrightarrow let\langle K\rangle\ p = M\ in\ E : K\ affirms\ C}\ \text{SAYS-E}$$

* Side condition: x must not occur in $\mathcal{P}$

Figure 1: Inference Rules

and that

$$\begin{aligned} M' \quad = \quad & \langle \mathsf{ACM}\rangle \\ & \quad let\langle \mathsf{ACM}\rangle\ p4 = p1\ in \\ & \quad let\langle \mathsf{ACM}\rangle\ p5 = p2\ in \\ & \quad\ aff\langle \mathsf{ACM}\rangle\ (p4\ \mathsf{Alice})\ ((p5\ \mathsf{Alice})\ p3) \end{aligned}$$

So the question is how Alice can convince ACM that there is a proof-term M making judgment (14) provable using the fact that judgment (15) is derivable. To do this, we observe that if Alice can demonstrate to ACM that each assumption in $\mathcal{P}_0$ is correct, then ACM will be convinced that judgment (14) is derivable. Think of it like this: judgment (15) says that from the hypotheses $\mathcal{P}_0$, ACM `says` `canDownload`(Alice) follows logically. If each hypothesis in $\mathcal{P}_0$ is itself valid, then ACM `says` `canDownload`(Alice) must be provable without any hypotheses.[4]

In summary, then, Alice must submit to ACM the following two pieces of evidence:

1. Evidence that the judgment $\mathcal{P}_0 \Longrightarrow M' :$ ACM `says` `canDownload`(Alice) is provable, and

2. Evidence that each policy in the hypotheses $\mathcal{P}_0$ is valid.

[4]Formally, this is called a substitution theorem. Such a theorem holds for all well-designed logics, including ours.

As we discuss later, the proof-term M' embedded in the judgment is enough evidence that the judgment is provable. So to prove (1), Alice only needs to submit the *judgment itself*. No additional evidence in needed. Evidence for (2) comes in two forms:

- Policies $p1$ and $p2$ are those of ACM itself. Hence ACM will know if they are valid or not. As such Alice does not have to submit any evidence to establish them.

- Policy $p3$: CMU `says` `isStudent`(Alice) comes from CMU. Hence ACM may not be aware of it a priori. To establish it, Alice must submit a *digitally signed certificate* containing the string "`isStudent`(Alice)". The certificate must be signed by CMU's private key. Once ACM verifies the signature, it is convinced that CMU actually believes that Alice is a student.

In general, evidence for hypotheses (policies) can be provided in one of the above forms: the reference monitor (ACM here) may be aware of some policies a priori; for these no evidence is needed. For others, the evidence comes in form of digitally signed certificates which must be submitted with the request.

This only leaves the question of showing that (1) can be established by the proof-term M' embedded in the judgment, i.e., that the proof-term suffices as evidence of its provability. We show a stronger result here. We show how the reference monitor can actually reconstruct Alice's proof from the proof-term M'. Having done this, it can check that the proof is correct, i.e., that each inference rule is correctly used. This is called *proof verification*.

Proof verification relies on one essential fact: given a proof-term constructor (like λ, Λ, aff etc.), there is *exactly one* rule whose conclusion contains a proof-term containing the constructor at the top-level. Hence by analyzing the top-level constructor of the proof-term, the reference monitor can determine the last rule used in the proof, and its exact premises. This can then be repeated on the premises (which have strictly smaller proof-terms). Continuing in this manner the entire proof can be reconstructed and checked. An essential observation is that this procedure requires no non-deterministic branching, and is linear in the size of the proof, hence very efficient. This procedure also justifies why fact 1 holds.

We illustrate proof verification by explaining some of its steps for our example. The reference monitor receives from Alice the judgment (15). Looking at the proof-term M', it observes that the top-level constructor is $\langle \mathsf{ACM} \rangle \ldots$. The only inference rule whose conclusion contains a proof-term of this form is SAYS-R. Therefore, this must have been the last rule in the proof. A little more analysis shows that the last inference rule must have been the following:

$$\frac{\mathcal{P}_0 \Longrightarrow E' : \mathsf{ACM}\ affirms\ \texttt{canDownload}(\mathsf{Alice})}{\mathcal{P}_0 \Longrightarrow M' : \mathsf{ACM}\ \texttt{says}\ \texttt{canDownload}(\mathsf{Alice})}\ \text{SAYS-R}$$

where

$$\begin{array}{rcl} E' & = & let\langle \mathsf{ACM}\rangle\ p4 = p1\ in \\ & & let\langle \mathsf{ACM}\rangle\ p5 = p2\ in \\ & & \quad aff\langle \mathsf{ACM}\rangle\ (p4\ \mathsf{Alice})\ ((p5\ \mathsf{Alice})\ p3) \end{array}$$

The top-level constructor of E' is $let\langle \mathsf{ACM}\rangle \ldots$. The only rule containing a proof-term of this form in its conclusion is SAYS-L. Thus this must have been the rule used to conclude the premise of the above rule. A little further analysis shows that the premise of the above rule must have been derived using the following rule:

$$\frac{\begin{array}{c}\mathcal{P}_0 \Longrightarrow p1 : \mathsf{ACM}\ \mathtt{says}\ \forall x.\ ((\mathsf{CMU}\ \mathtt{says}\ \mathtt{isStudent}(x)) \supset \mathtt{isStudent}(x)) \\ \mathcal{P}_1 \Longrightarrow E'' : \mathsf{ACM}\ affirms\ \mathtt{canDownload}(\mathsf{Alice})\end{array}}{\mathcal{P}_0 \Longrightarrow E' : \mathsf{ACM}\ affirms\ \mathtt{canDownload}(\mathsf{Alice})}\ \text{SAYS-L}$$

where

$$\begin{array}{rcl} E'' & = & let\langle \mathsf{ACM}\rangle\ p5 = p2\ in \\ & & \quad aff\langle \mathsf{ACM}\rangle\ (p4\ \mathsf{Alice})\ ((p5\ \mathsf{Alice})\ p3) \end{array}$$

This process can now be continued on the two premises, and the entire proof constructed in section 4.2 can be reconstructed backwards. Having done this, ACM can now check that each rule was correctly applied.

In summary, in proof-carrying authorization, the requester submits a logical judgment (with a proof-term) stating that it is allowed access and evidence that its hypotheses are valid (possibly in the form of digitally signed certificates). The reference monitor reconstructs the requester's proof from the proof-term, verifies that it is correct and that the hypotheses are valid. If all these succeed, the reference monitor allows access, else denies it. This is extremely efficient because it is usually straightforward for the requester to construct a proof, and for the reference monitor to verify it. In contrast, approaches requiring the reference monitor to determine whether access should be allowed (or not) without help from the requester are inefficient since the reference monitor is not aware of the reason why this is the case. It must generically reason with an inefficient and often incomplete algorithm to make this decision.

References

[1] Martín Abadi. Access control in a core calculus of dependency. In *ICFP '06: Proceedings of the eleventh ACM SIGPLAN international conference on Functional programming*, pages 263–273, New York, NY, USA, 2006. ACM Press.

[2] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[3] Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In G. Tsudik, editor, *Proceedings of the 6th Conference on Computer and Communications Security*, pages 52–62, Singapore, November 1999. ACM Press.

[4] Lujo Bauer. *Access Control for the Web via Proof-Carrying Authorization*. PhD thesis, Princeton University, November 2003.

[5] Lujo Bauer, Scott Garriss, Jonathan M. McCune, Michael K. Reiter, Jason Rouse, and Peter Rutenbar. Device-enabled authorization in the Grey system. In *Information Security: 8th International Conference (ISC '05)*, Lecture Notes in Computer Science, pages 431–445, September 2005.

[6] Lujo Bauer, Scott Garriss, and Michael K. Reiter. Distributed proving in access-control systems. In *Proceedings of the 2005 Symposium on Security and Privacy*, pages 81–95, May 2005.

[7] 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*, August 2002.

[8] Deepak Garg and Frank Pfenning. Non-interference in constructive authorization logic. In J. Guttman, editor, *Proceedings of the 19th Computer Security Foundations Workshop (CSFW '06)*, pages 283–293, Venice, Italy, July 2006. IEEE Computer Society Press.

[9] 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.

[10] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. Delegation logic: A logic-based approach to distributed authorization. *ACM Trans. Inf. Syst. Secur.*, 6(1):128–171, 2003.

[11] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[12] Andrew Pimlott and Oleg Kiselyov. Soutei, a logic based trust-management system. In *Proceedings of the Eighth International Symposium on Functional and Logic Programming*, 2006.