Information Carrying Identity Proof Trees

Abstract

In open distributed systems, the verification of properties of subjects is a crucial task for authorization. Very often access to resources is based on policies that express (possibly complex) requirements in terms of what are referred to variously as identity properties, attributes, or characteristics of the subject. Example characteristics include whether the subject is (operating on behalf of) a user of a certain age or having a certain credit rating, or is an organization having certain accreditation, to name just a few. In a distributed system having no central authority on subject characteristics, evaluation of such policy requirements is a challenging task. In this paper we provide an approach according to which an entity, referred to as verifier, can evaluate a query concerning properties related to the identity of a subject, which may be required for the purpose of authorizing some action. The present contribution concerns the reuse of query results. We consider issues related to temporal validity (*i.e.*, expiration and revocation of identity properties) as well as issues related to confidentiality when one entity reuses query results computed by another entity. We employ constraint logic programming as the foundation of our policy rules and query evaluation. This provides a very general, flexible basis, and enable our work to be applied more or less directly to several existing policy frameworks. The process of evaluation of a query against a subject identity is traced through a structure, referred to as identity proof tree, that carries all information proving that a policy requirement is met.

1 Introduction

In open distributed systems, the verification of identity properties of subjects is a crucial task for access control. Very often access to resources is based on policies that are expressed in terms of (possibly complex) conditions against such identity properties. The evaluation of such conditions in a distributed system in which there is no central entity recording all relevant information is a challenging task. In many cases, the *relying party*, that is, the party providing a service and thus having to verify access control policies for service provisioning, may have to delegate to other parties in the system, referred to as *verifiers*, the verification of (some of) the policy conditions. Delegation may be required for several reasons. For example, the information needed to verify conditions against certain identity properties of a subject is owned by a party different from the relying party; the former party may not be willing to transmit such information to the relying party but may be willing to perform the verification. Notice that a verifier may in turn delegate to other verifiers the execution of some portions of its verification tasks. Therefore, the verification process can be quite articulated and involve several verifiers, some of which may not even have direct relationships with the relying party. Current standards such as SAML support the interchange of assertions about subjects among the parties in a system. It is however important to notice that such standards only convey minimal assertions, such as for example the so-called "attribute assertions" that simply state the values of identity properties of subjects. We believe that it is important to provide the relying party, which ultimately has to make a decision whether or not to grant access to a given resource, with a rich set of information about the verification process that has taken place. Relevant information include the policies that a verifier has applied in order to verify a certain condition, temporal validity of a verification, credentials used by a verifier. We can consider this set of information as the "provenance" information concerning the verification of conditions against some identity properties of a given subject.

The goal of the work reported in this paper is to develop the notion of *identity proof tree*, that is, a structure encoding all information used during the process of proving conditions against the identity properties of a given subject. Some relevant requirements of an approach to the definition and management of identity proof trees include the following:

- **Policy Compliance.** The structure should convey all necessary information to prove that a policy requirement is met. It should be possible to check the validity of this structure without needing to search among possible partial proofs for one that can be completed. However re-verification of revocation status of properties in the structure may be needed if significant time has elapsed since the verification was last performed.
- **Verification Reuse.** It should be possible to reuse verifications of properties performed by other verifiers. If properties verified by others are combined with one another or with properties the relying party validates itself, we need to have a means to organize the reliance on the other verifiers. The level of assurance obtained from properties verified by others must be assessed, recorded, and managed.
- **Timeliness.** Checking expiration and revocation of credentials used in a derivation is an important aspect when relying on verifications performed by other parties. An attestation concerning the verification of a property must have associated some temporal validity information. In particular, it should include information sufficient to enable the relying party to check whether any of the properties have been revoked.
- **Confidentiality and privacy.** It should be possible for a verifier to maintain confidentiality of personal information used in the verification of a property. In particular, a verifier can share the result of a verification with a relying party that the relying party may not be able to verify for herself, thereby facilitating an increase in legitimate utilization. However, care must be taken to ensure that the very fact that the authorization policy is satisfied does not disclose information in violation of privacy requirements.

In the current discussion, we use standard notions from constraint logic programming (CLP) to formalize policy requirements and rules for their derivation. This enables us to handle easily such needs as chaining logical implications, as well as instantiation of and constraints on variables. However, as we return to in the final section, this is not intended to be a concrete representation for exchange of identity proof trees, and a suitable one is certainly needed.

The main contributions on this work can be summarized as follows. We introduce the use of a proof tree annotated with several kinds of information, based on which we provide formal techniques to verify the validity of individuals' properties. We support decentralization of proof verification, in that we do not require a single verifier to complete the process of verification, but we allow multiple entities to cooperate in the process verification. We provide techniques verifiers can use to control sharing derived results that reflect information about sensitive personal properties that enables participants to avoid disclosing a level of detail about such information not authorized by the information owner. We enable reuse of verifications of derived characteristics by incorporating temporal values in the representation of such verifications that specify the times at which the derived characteristic is valid. Finally, by using CLP, policies can be very expressive, while specifying unambiguously and flexibly the amount of information that is authorized to be disclosed. While policies permitting release of partial information about personal properties have been proposed in the past [12], the current proposal is much more flexible. Arbitrary constraints can be used to state the degree of specificity a given class of recipients is authorized to receive about a given personal property.

The rest of the paper is organized as follows. In the next section we provide an overview of our approach. In Section 3.2 we survey some basic notions related to constraint DATALOG. In Section 5 we elaborate on the notion of Identity proof tree, and on its properties. In Section 6 we discuss techniques to prevent sharing derived results that cause undesired leak of information. In Section 7 we analyze related work, while in Section 8 we conclude the paper.

2 Approach

In our approach, authorization is based on properties of entities requesting resources, properties that we call *characteristics* and which we formalize by using logical atomic formula. A characteristic is a property that an entity, called the characteristic's *subject*, is deemed to have by another entity, called the characteristic's *issuer*. It may be a compound property, comprising many aspects, much as a driver license comprises many attributes, such as age, class of vehicle the driver is authorized to operate, the driver's height, eye color, etc. It also is intended to be quite general. For instance, one's affiliations and roles in those organizations, one's licenses or degrees held, nationality, and medical status are all examples of properties that could form characteristics in our system.

A characteristic that is asserted by its issuer can be placed in a credential, thereby making its authenticity verifiable, typically by means of public key signatures. A rule for deriving a characteristic, which is formalized as a logical clause, can also be placed in a credential, thereby also making verifiable its authenticity and integrity as an assertion by the issuer of the derived characteristic.

In this work we propose techniques to enable the verification of an individual's characteristics to be reused, either by the verifier herself or by another relying party. We do this with a formal policy system based on an efficient subset of constraint logic programming, DATALOG with constraints, which we introduce in the next section.

Two key issues arise when verifications are reused, particularly by a third party. First, the characteristics on which the verification is based are often transient. For instance, a principal's employment relationship may terminate, expectedly or otherwise. Thus, the timeliness of characteristics is one central concern.

Another is that characteristics are potentially sensitive. In our system, individuals can define what we call characteristic release policies (see section 4), which specify authorized recipients of each of the entity's characteristics, as well as the degree of precision about the characteristics that the recipient is authorized for. For instance, Alice might be willing to tell her colleagues that her salary is between \$50K and \$150K, but the general public would only be authorized to know that it is between \$15K and \$200K.

Our techniques are based on the use of a data structure, introduced below in section 5, that we call an identity proof tree (IpT) and that represents a formal proof of a characteristic of a principal. *By allowing identity proof trees to be summarized as logical clauses, and enabling verifiers of these trees to share the resulting clauses with others who may not be authorized for all the information in the proof tree, it becomes possible for service providers to grant requests they could not without the verifier's assistance.* We present techniques a verifier can use to ensure that a derived characteristic it releases to a relying party does not leak information for which the recipient is not authorized.

Derived characteristics can be saved and reused if temporal validity of the characteristics can be ensured. We support ensuring this by providing techniques that manage expiration and revocation of characteristics.

We leverage constraints to reason about time by embedding a current time variable in the characteristics and by including temporal constraints on the clauses, to trace the time during which the characteristic is valid. This aspect is of particular relevance in case of large distributed domains, where availability of up to date status information is not always possible. We thus allow multiple usage of successful validation. Assurance of the authenticity and currentness of clauses for deriving characteristics can be provided by placing the clauses in digital credentials cryptographically signed by the principal that asserts that the characteristic holds when properly derived. This principal is called the *issuer* and is the only entity able to create, modify, or remove (revoke) the credential or the clause that it carries. The cryptographic signatures provide high assurance of this for a period of time. However, the resistance of such signatures to forgery decreases as time elapses. To mitigate this threat credentials typically expire after a limited time. Fresh credentials carrying the same clauses can then issued as needed using a fresh key. The expiration date expresses the longevity of the credential as evidence of the characteristic. Additionally, credentials may be revoked before the stated expiration date, for instance, if the signing key is compromised or the content of the credential is no longer valid. Determining credentials' temporal validity status is not trivial in distributed environments, since principals may not always have access to the most up to date status information on credentials' validity.

Like credentials, characteristics have lifetimes, though they often do not coincide with the lifetimes of the credentials that carry them. A characteristic's lifetime is the time period during which the subject enjoys the property given by the characteristic. For instance, the validity period of an employment characteristic ends when the employer/employee relationship is severed, not when the employee requires a new credential documenting the relationship. Only the lifetime of the characteristic is reasoned about within the clauses that compose the authorization policy in this paper. Checking the validity of the credentials that carry these clauses needs to be done when the IpT is initially constructed and need not be modeled within the policy system itself. This is because it serves to support the secure communication of the policy, which is not a concern of the policy itself.

3 Preliminary notions

This section presents the subset of constraint logic programming that we use to express our authorization policies. It begins by giving a brief overview of DATALOG with constraints, and then shows how we express entity properties within this formalism.

3.1 Brief overview of Constraint Datalog

Constraint logic programming (CLP) extends classical logic programming (LP) based on Horn clauses by adding constraints expressed over predefined domains. So while classical LP relies on predicates and functions defined entirely within the program, CLP puts at the programmer's disposal relations and operations defined over a predefined domain. This domain is parametric. In this section we summarize the subset of CLP that we use to express our authorization policies.

DATALOG is a restricted form of LP which has variables, constants, and predicate symbols, but no function symbols. A DATALOG *clause* has the form $p_0(\mathbf{t}_0) \leftarrow p_1(\mathbf{t}_1), \ldots, p_n(\mathbf{t}_n)$ in which, for each i, $1 \leq i \leq n$, $\mathbf{t}_i = t_{i,1}, \ldots, t_{i,k_i}$ is a vector of terms, *i.e.*, variables and constants, and p_i is a k_i-ary predicate symbol. The atomic formula $p_0(\mathbf{t}_0)$ is called the *head* and the sequence $p_1(\mathbf{t}_1), \ldots, p_n(\mathbf{t}_n)$ is called the *body*. A clause is *safe* if all variables occurring in the head also occur in the body. If the body is empty ($n = 0$), the clause is called a *fact*; otherwise, it is called a *rule*. A DATALOG program is a finite set of safe DATALOG clauses. We use H and L to denote atomic formulas (usually called simply *atoms*, possibly with subscripts.

Advantages of using DATALOG as a policy language include: (1) its semantics are that of logical entailment which are unambiguous and clear; (2) the absence of function symbols ensures that queries can be evaluated in time that is a small polynomial of the number of clauses in the policy; (3) there are many efficient, well understood methods of evaluation.

In this paper we use DATALOG extended with constraints [10], denoted DATALOG$^{\mathcal{C}}$, as the language in which authorization policies are expressed. When a formula is found to hold under a given constraint, the meaning is that it holds under any valuation of the variables that satisfies the constraint. This often enables a concise specification of a large collection of satisfactory values.

We now overview key notions of DATALOG$^{\mathcal{C}}$. Intuitively, a constraint domain is a domain of objects, such as numbers or files in a directory, together with a language for speaking about these objects. The language is typically defined by a set of first order constants, function symbols and predicate (relation) symbols.

Definition 3.1 A *constraint domain* Φ is a 3-tuple $(\Sigma, \mathcal{D}, \mathcal{L})$. Here, Σ is a signature; it consists of a set of predicate symbols and a set of function symbols, each with an associated "arity", indicating the number of arguments to the symbols. Zero-ary function symbols are called constants. $\mathcal{D}$ is a Σ-structure; it consists of (1) a set D called the universe of the structure, (2) a mapping from each constant to an element in D, (3) a mapping of each k-ary predicate symbol in Σ to a k-ary relation over $\mathcal{D}$, and (4) a mapping of each k-ary function symbol to a function in $\mathcal{D}^k \rightarrow D$. Atomic formula formed using Σ and a countably infinite set of variables V are called *primitive constraints*. $\mathcal{L}$ is a class of quantifier-free-first-order formulas over Σ, which are called the *constraints* of Φ.

For example, given the signature consisting of integer numerals and the arithmetic operators and comparison operators, together with the integers and the customary interpretation of the operators, $\langle > (+(x, y), 4)\rangle$ is a primitive constraint which is almost always written in the usual infix concrete syntax as $x + y > 4$.

The domain of real numbers is one of several powerful constraint domains studied in the literature. It is straightforward to encode open mathematical questions (such as Fermat's conjecture) and uncomputable functions by using it. Naturally in our context we need to ensure that evaluation is efficient, so we will limit the power of the domains we use. Several domains that are useful for authorization policies are shown to support tractable evaluation in [14]. This includes a domain whose basic constraints require that real-valued variables lie in constant-bounded intervals, as would support checking whether, say, age or income lie within certain ranges.

Definition 3.2 In DATALOG$^{\mathcal{C}}$, formulas contain two sorts of variables: constraint-domain variables (or just *domain variables*) and DATALOG variables. These sorts are disjoint. A DATALOG$^{\mathcal{C}}$*clause* has the form $H \leftarrow L_1, \ldots, L_n; \phi$, where $H \leftarrow L_1, \ldots, L_n$ is a DATALOG clause, possibly containing some domain variables, and ϕ is a constraint over the domain variables occurring in it. When $n = 0$, the clause is a DATALOG$^{\mathcal{C}}$ *fact*, $H; \phi$. A DATALOG$^{\mathcal{C}}$ *query* Q has the form $L_1, \ldots, L_m; \phi$ and consists of one or more atoms together with a constraint. A DATALOG$^{\mathcal{C}}$ program is a finite set of DATALOG$^{\mathcal{C}}$ clauses.

The assertion of a clause $H \leftarrow L_1, \ldots, L_n; \phi$ signifies that for all valuations θ of the domain variables that satisfy ϕ, $\forall \mathbf{x}(\theta(H \leftarrow L_1, \ldots, L_n))$ holds, in which $\mathbf{x}$ enumerates the DATALOG variables in the clause. For example, the clause$L_1(x, y) \leftarrow L_2(x), L_3(y); x + y > 4$ expresses that $L_1(x, y)$ holds if $x + y$ is greater than four and both $L_2(x)$ and $L_3(y)$ are true.

A query $L_1, \ldots, L_m; \phi$ asks for a solution of the form $L'_1, \ldots, L'_m; \psi$ such that $L'_1, \ldots, L'_m$ is obtained from $L_1, \ldots, L_m$ by substituting DATALOG terms for DATALOG variables, and ψ logically entails ϕ (*i.e.*, $\psi \models \phi$). Notice that the clauses in a DATALOG$^{\mathcal{C}}$ program are not necessarily safe. This is because the atomic formulas derived from DATALOG$^{\mathcal{C}}$ programs can contain variables that are subject to constraints.

We restrict our attention to constraint domains Φ that admit existential quantifier elimination, meaning that every formula formed by existentially quantifying a constraint can be rewritten to an equivalent quantifier-free constraint. In fact we will only consider domains for which this rewriting is effective and tractable.

3.2 Formal language for specifying characteristics

In this section we explain how principals' characteristics and authorization policies are specified in our model, which is done in both cases by using DATALOG$^{\mathcal{C}}$ clauses. We first briefly describe a few assumptions about the infrastructure on which our system depends.

We assume the existence of nonempty sets $\mathcal{S}$ of principals and $\mathcal{O}$ of objects, and a finite, nonempty set of actions $\mathcal{A}$ executable on objects in $\mathcal{O}$. Service providers are principals that control access to objects in $\mathcal{O}$. Objects can be logically organized into hierarchies and accessed by performing specific actions in $\mathcal{A}$.

We use i, r and s to denote terms (constants or variables) with values in $\mathcal{S}$ and a for terms in $\mathcal{A}$. We use x, y, u, w, and z for variables ranging over arbitrary domains (not just principals), and t to denote general terms. We use $\mathbf{t}$ and $\mathbf{x}$ to represent tuples of terms and variables, respectively.

3.2.1 Characteristics

A characteristic is given by $\langle p(\mathit{IDs}, s, i, vt, \mathbf{t}); \phi\rangle$ in which $p(\mathit{IDs}, s, i, vt, \mathbf{t})$ is an atomic formula that has a number of mandatory, predefined parameters, and ϕ is a constraint over the variables that appear in the atomic formula. We presently discuss *IDs*, the first of the mandatory parameters. Other mandatory parameters in the formula are the *subject* s, *i.e.*, the entity being characterized, the characteristic's *issuer* i, and the *valid time* vt. Subject and issuer are principals, while valid time takes on time values at which the characteristic holds.

The user defined parameters, given by $\mathbf{t}$, typically includes, among the other things, two *temporal bounds*, expressing the beginning t_b and the end t_e of the period during which the characteristic is valid. These values are represented as *date expressions* of the form `mm/dd/yy:hh`. The value of t_e can also be ∞, with the obvious intended meaning.

The first of the predefined parameters, *IDs*, is a list of serial numbers of clauses that can be used to derive the atomic formula. Such serial numbers can be used to uniquely identify clauses that have been revoked. Thus the list of serial numbers can be used to determine whether a previously verified characteristic has become invalid do to a revocation. The list is constructed by using a programming idiom common in Prolog called *difference lists* [19]. We do not present the details of this technique here, but instead simply illustrate its use, which requires some care. The benefit of using difference lists instead of standard list manipulation primitives, such as append, is that they enable us to ensure that the cost of constructing the list is linear in the number of clauses used to derive a given characteristic. Although space prevents our presenting it here, it is straightforward to automate the introduction of this parameter. Thus its somewhat technical nature should not concern the reader.

IDs is the only place in our policy language that function symbols are permitted: list constructors are used to enable this parameter to report clauses whose revocation would call into question the validity of the characteristic's derivation. So long as this parameter is managed in the manner illustrated below, the presence of function symbols here does not jeopardize tractability.

Example 1 *Bob requests truck-rental service from U-Truck, a company specialized in renting large vehicles that requires all drivers to be covered by specific accident insurance. U-Truck contracts with the Rock Insurance Company for this coverage, which provides several different policy riders, based on renters' experience and driving record. Also, if Bob is proven to be an experienced driver and that he has never been involved in car accidents, a special rate is offered. If the applicant is proven to be a good citizen then a separate discount is also available.*

Bob meets the above requirements, and is already a customer of Rock Insurance (RockIns), which underwrites his personal car insurance. The characteristic

$$\langle \text{ DriverLicence(diff([1250249],[]), Bob, DMV, } vt, t_b, t_e, \text{DoB}); \\ t_b \leq vt \wedge vt \leq t_e \wedge \text{DoB} = 10.11.80, t_b = 04.01.99{:}16 \wedge t_e = 10.11.06\rangle$$

says Bob was born on the 11th of October, 1980. Bob was issued the driver license at 16:00h on 1 April 1999 by the Department of Motor Vehicles, which is valid until his birthday in the year 2006. The serial number of this clause is 1250249. We use Prolog style list specification; [1250249] represents a list whose sole element is the serial number 1250249 and [] represents the empty list. The function symbol, diff, is an unevaluable constructor that serves to group the pair of lists that form the difference list. Through the characteristic

$$\langle \text{ MD(diff([2393838],[]), Bob, JohnsHopkins, } vt\text{); } 12.16.04{:}10 \leq vt\rangle$$

JohnsHopkins asserts that Bob has a MD degree. The characteristic is valid ever since the degree was conferred on the 16th of December 2004, and it does not expire.

3.2.2 Policy statements

A policy statement in our language is a DATALOG$^{\mathcal{C}}$ clause in which the atomic formulas are characteristics[1]. We use $\mathcal{P}$ to denote the set of DATALOG$^{\mathcal{C}}$ clauses representing policy statements that are valid (*i.e.*, issued, but neither expired nor revoked) at a given point in time.

[1]The focus of the current work is not concerned with HCI aspects of policy capture, so we allow arbitrary DATALOG$^{\mathcal{C}}$ clauses.

Example 2 *The only entity that could create the following clause defining MD would be Johns Hopkins Medical School:* MD(diff([2393838 | Tail], Tail), Bob, JohnsHopkins, vt); $12/16/04{:}00 \leq vt$. *(The notation* [2393838 | Tail] *denotes a list whose first element is the number* 2393838 *and whose remaining elements are given by the logical variable,* Tail. *So if* Tail *becomes constrained to have a specific list as its value, this list will give the remaining elements of* [2393838 | Tail].*)*

Similarly, the only entity that could create the following clause defining eligibility to receive a Driver License would be one that speaks for the authority of Department of Motor Vehicles:

$$\begin{aligned}&\mathrm{DriverLicenceApplication(diff([4721993 \mid Tail1], Tail4), s, DMV,}vt\mathrm{, DateofBirth)} \leftarrow\\ &\quad \mathrm{BirthCert(diff(Tail1, Tail2), s, h,}vt\mathrm{, DateofBirth),}\\ &\quad \mathrm{regiseredHospital(diff(Tail2, Tail3), h, USFedCode,}\ vt\mathrm{), Passport(diff(Tail3, Tail4), s, US,}\ vt);\\ &\quad \mathrm{DateofBirth} \leq vt - (16 * 365).\end{aligned}$$

A query issued on April 1, 1999 might look like ⟨DriverLicenceApplication(diff(*SerialNumbers*,[]), Bob, DMV, vt, DateOfBirth); $vt = 04.01.99$⟩.

The clause above could be used to derive the following solution:

$$\langle\ \text{DriverLicenceApplication(diff([472199,4324693,5382944,4530247],[]), Bob, DMV,}\\ vt, 10.11.80); vt = 04.01.99\rangle$$

when provided additional clauses such as:

$$\begin{aligned}&\mathrm{Passport(diff([4530247 \mid Tail], Tail), Bob, US,}vt); 01.01.99{:}19 \leq vt \wedge vt \leq 13.08.06{:}07.\\ &\mathrm{BirthCert(diff([4324693 \mid Tail], Tail), Bob, ChicagoGeneral,}vt\mathrm{, 10.02.80{:}19)}; 10.02.80{:}19 \leq vt.\end{aligned}$$

and a derivation of registeredHospital(diff([5382944 | Tail],Tail), ChicagoGeneral, USFedCode).

Notice that the temporal validity of Driver License will be established by DMV, but it will possibly be influenced by constraints on the temporal validity status of the clauses used to determine the Driver License.

As mentioned above, the management of the list of serial numbers in the first parameter can be performed automatically, though space limitations prevent our presenting this here. In the remainder of this paper, with the exception of section 5.3 where managing revocation is discussed, we omit this parameter from example clauses, so as to assist the reader in focusing on more relevant aspects of the examples.

Policies are used to control access to objects and to express conditions against the identity properties of the subjects. In the context of DATALOG$^{\mathcal{C}}$, this is done in two parts that are independent and not necessarily ordered. The first part is concerned with the definition of user characteristics. In it, issuers of characteristics write DATALOG$^{\mathcal{C}}$ clauses defining the predicates used to express those characteristics. Clauses with empty body (*i.e.*, facts) are used to express identity properties of specific individuals. One the other hand, clauses with non-empty bodies are rules used express general principals according to which a characteristic can be derived from one or more other characteristics. and those clauses express conditions against the identity properties of the subjects. The second part is concerned with defining the characteristics that are necessary for authorizing access to specific objects. A relying party associates with each protected object under her control a combination of characteristics sufficient to authorize access to the object. Note that the relying party may be the issuer defining the characteristics that grant access, or she may delegate this authority to others.

Example 3 *Suppose Rock Insurance needs to have assurance of Bob's reliability and trustworthiness. Thus, it specifies a policy stating that a citizen is reliable if he is accredited as a good citizen from AuditOrg*[2]*. The clause below is from AuditOrg, which states that in order to be considered a GoodCitizen the principal needs to have a good credit record and no felony record.*

$$\begin{aligned}&\mathrm{GoodCitizen(s, AuditOrg,}vt, t_{gb}, t_{ge}) \leftarrow \mathrm{NoFelonyRecord(s, Org,}vt),\\ &\quad \mathrm{CreditRecord(s, Experion,}vt, \mathrm{tBegin}_{PT}, \mathrm{tEnd}_{PT}, \mathit{Rating}); 600 \leq \mathit{Rating}, \mathrm{tEnd}_{PT} < t_{ge}, \mathrm{t}_{bg} \geq vt, \mathrm{t}_{be} \leq vt\end{aligned}$$

[2]This would be an auditing company affiliated with the insurance company

Rock Insurance offers special insurance deals for cars rented to medical doctors who graduated within the past 2 years.

$$\begin{aligned}&\mathrm{SpecialOffer(van, s)} \leftarrow \mathrm{GoodCitizen(s, AuditOrg,}vt, \mathrm{t}_{gb}, \mathrm{t}_{ge}),\\ &\quad \mathrm{MD(s, JohnsHopkins,} vt, MD\_Date); \mathrm{t}_{gb} \leq vt - (2 * 365), \quad MD\_Date \leq \mathrm{t}_{gb}.\end{aligned}$$

The expression $t_{gb} \leq vt - (2 * 365)$ poses a time constraint on the temporal bounds of the GoodCitizen characteristic, requiring that it have been issued in the last two years. The constraint on the MD characteristic requires the principal to have received the MD degree after the issue of the GoodCitizen characteristic, and thus in the last two years.

4 Characteristic Release Policies

In general, principals may consider their characteristics to be sensitive, and wish to share them only with certain entities. Of course, different characteristics may be sensitive to different degrees, but also the level of detail provided about a characteristic may affect how sensitive a disclosure of the characteristic is considered to be.

We now introduce a syntactic structure called an *characteristic release policy* (CRP), which enables principals to specify which entities they permit to receive which characteristics and with what level of detail about attribute (parameter) values.

Definition 4.1 A Characteristic Release Policy (CRP) consists of a set of statements (called *disclose statements*) each having the following form:

$$\textit{disclose}\ \langle p(s, i, vt, \mathbf{t}); \phi\rangle\ \textit{if}\ \langle q_1(r, i_1, vt, \mathbf{t}_1), \ldots, q_m(r, i_m, vt\mathbf{t}_m); \psi\rangle\ \textit{for <purpose-list>}$$

in which $\mathbf{t} = t_1, \ldots, t_n$ and $\mathbf{t}_j = t^j_1, \ldots, t^j_{n_j}$, in which n_j is the arity of q_j, s is the subject of the characteristic, i is the issuer, and, additionally, r is a variable denoting the would-be recipient of the information and the i_j's are issuers of his characteristic. If the query given by $\langle q_1(\mathrm{Bob}, i_1, \mathbf{t}_1), \ldots, q_m(\mathrm{Bob}, i_m, \mathbf{t}_m); \psi\rangle$ can be satisfied, then Bob is authorized for the purpose of actions listed in <*purpose-list*>. A characteristic can be made available to all service providers by making $m = 0$. If there is no statement for a given characteristic, it cannot be disclosed to anyone.

The actions listed in the <*purpose-list*> can be requested from a service provider, who will use it to determine the label of the root node in any IpT that shows authorization to have the action performed. Intuitively, a statement of this form means that any entity r from which the principal requests an action in <*purpose-list*> and that satisfies $\langle q_1(r, i_1, \mathbf{t}_1), \ldots, q_n(r, i_m, \mathbf{t}_m); \psi\rangle$ is authorized to know that there exists a valuation θ of the variables occurring in $p(s, i, \mathbf{t})$, that satisfies ϕ and $\mathcal{P} \models \theta(p(s, i, \mathbf{t}))$. When the purpose list is empty, the disclose statement authorizes disclosure for any purpose.

Example 4 *To satisfy U-Truck's requirements, Bob must acquire an insurance policy from Rock Insurance, which applies different policies according to age, which has to be in between 25 and 65.*

Suppose that Bob is willing to disclose his driver license for renting the truck to any rental agency that is part of the American Rental Organization. In the disclose statement shown below, Bob states that he is prepared to have it disclosed to such an agency that according to his driver license, his age is between 25 and 60. Note however that it does not authorized disclosure of Bob's precise age.

$$\begin{aligned}&\textit{disclose}\ \langle \mathrm{DriverLicence(Bob, DMV,}vt, DateofBirth, t_b, t_e);\\ &\quad DateofBirth < vt - (25 * 365), DateofBirth > vt - (60 * 365),\\ &\quad t_b \leq vt, vt \leq t_e, t_b = 04.01.99{:}19, t_e = 10.11.06\rangle\\ &\quad \textit{if}\ \langle Rental\_Insurance\_Rep(r, AmericanRentalOrg); \textit{true}\rangle\\ &\quad \textit{for}\ \langle Car\_Rental, Truck\_Rental, MiniVan\_Rental\rangle\end{aligned}$$

In section 6, we present techniques by which a verifier, trusted by Bob with the full details of Bob's age and trusted by a relying party to faithfully verify that information, can provide the relying party with validation of characteristics of Bob that depend on sensitive characteristics in a manner that complies with the limits of detail about the sensitive characteristics specified in disclose statements. In other words, the verifier will assure the relying part that Bob's sensitive characteristics satisfy the relying party's policy requirements just in case doing so cannot disclose information about Bob that would violate Bob's disclosure policy. While techniques for enabling Bob and the relying party to establish this sort of mutual assurance by using secure multiparty computation [11] have previously been proposed, such techniques are rather heavyweight. The techniques presented here are much simpler and likely to be much more efficient in settings where mutually trusted third party verifiers are not difficult to find.

5 Identity Proof Trees

In this section we introduce the notion of Identity Proof Tree, a structure encoding all information used in the proof that required conditions are met by the identity properties of a given subject.

5.1 Identity Proof Tree

We assume that for each entity and each point in time, there is a well defined set of all valid (neither expired nor revoked) credentials in the system that are available to the entity at that time. We assume that the assignment of such sets is well behaved with respect to the sets seen by other entities and at other times. Certain measures can be taken to make this reasonable, such as ensuring that once a credential is revoked it cannot be reissued. (Instead a new credential with a new serial number can be issued carrying the same characteristic, if need be.) The *current policy state* $\mathcal{P}$ is defined to be the clauses carried by members of this set of valid credentials.

We organize the derivation of a characteristic from the current policy state $\mathcal{P}$ in a structure that we call an identity proof tree (IpT). In practice, IpTs form the basis for making authorization decisions. In fact, to evaluate a query for an action a on object o, we check whether or not an identity proof tree for a query can be constructed from $\mathcal{P}$. Identity proof trees are formally defined as follows.

Definition 5.1 Let $\mathcal{P}$ be a DATALOG$^{\mathcal{C}}$ program defining the current global policy state. An *identity proof tree based on* $\mathcal{P}$ $\mathcal{T}$ =$\langle \mathcal{R}, \mathcal{N}, \mathcal{E} \rangle$ is a finite tree satisfying the following properties:

- $\mathcal{N}$ (the set of nodes). Each node $n \in \mathcal{N}$ is a triple $n = \langle \mathit{UpL}, \mathit{LoL}, \mathit{Const} \rangle$ in which:
  - *UpL* is an atomic formula called the *upper label of* n;
  - *LoL*, is either an atomic formula or $\bot$, and is called the *lower label of* n;
  - *Const* is a constraint. If $\mathit{LoL} = \bot$, $\mathit{Const} = \mathit{true}$. Otherwise, there is a renaming variant of a clause $H \leftarrow L_1, \ldots, L_\ell; \phi$ in $\mathcal{P}$, such that $\mathit{LoL} = H$ and H has the same predicate symbol as *UpL*, and, letting *UpL* is $p(s_1, \ldots s_m)$ and $\mathit{LoL} = H = p(t_1, \ldots t_m)$, $Const \equiv s_1 = t_1 \wedge \cdots \wedge s_m = t_m \wedge \phi$. There is one exception: the constraint labeling the root can be stronger. (See below.)

  We select the components from a given node n by writing $n.\mathit{UpL}$, $n.\mathit{LoL}$, and $n.\mathit{Const}$.

- $\mathcal{R} = \langle \mathit{UpL}, \mathit{LoL}, \mathit{Const} \rangle$ is the root of the tree. The constraint labeling the root is the same as for other nodes, except that an additional constraint can be conjoined to it, representing a constraint on the characteristic in a query.

- $\mathcal{E}$ (the set of directed edges). Each $e \in \mathcal{E}$ is an ordered pair $e = \langle n_1, n_2 \rangle$, with $n_1, n_2 \in \mathcal{N}$ representing that n_1 is the *parent* of n_2 and n_2 is a *child* of n_1.

For each node n such that $n.\mathit{LoL} \neq \bot$, there exists a renaming variant $H \leftarrow L_1, \dots, L_\ell; \phi$ of some clause in $\mathcal{P}$, such that $n.\mathit{LoL} = H$ and n has exactly ℓ children whose upper labels are $L_1, \dots, L_\ell$, respectively.

The root of an identity proof tree has as its upper label a characteristic, which is typically a characteristic that authorizes to some action. If the lower label of a node n is not $\bot$, n is said to be *closed*. Note that the constraint labeling a node can refer to variables that do not appear in the upper or lower label of the node, but rather in the upper label of one of the node's children. The clause variants used in the tree must not share any variables with one another and the set of constraints labeling nodes in the tree must be consistent (*i.e.*, satisfiable). The tree $\mathcal{T}$ is consistent if there is a valuation of variables that satisfies all constraints in the tree.

When an IpT $\mathcal{T}$ contains only closed nodes, we call $\mathcal{T}$ *closed*. In this case, $\mathcal{T}$ represents a complete proof that the upper label of the root holds for all valuations of the variables that satisfy all constraints in $\mathcal{T}$. More precisely, the upper label of the root holds for all valuations that satisfy the constraint $\overline{\exists}\mathbf{x}.\ \wedge_{n \in \mathcal{T}.N}\ n.Const$, in which the tuple $\mathbf{x}$ comprises the variables appearing in the upper label of the root[3]. Given an action a, an object o, and a principal s, if a relying party verifies the correctness of a closed IpT $\mathcal{T}$ based on $\mathcal{P}$, then she knows that s is authorized for action a on object o.

5.2 Decentralized Identity Proof Trees

We now consider how a relying party can gain reasonable assurance that s is authorized for a by making use of verifications performed by others and without having to verify the correctness of the entire IpT herself. In the simplest case, this is achieved if a verifier, trusted by the relying party, checks that such an IpT exists, and simply reports to the relying party that s satisfies the characteristic. To generalize this, notice that a large closed IpT can be viewed as being composed of several smaller IpTs, some of which may not be closed. Two IpTs can be combined into one if a non-closed node in one has the same upper label as the root of the other and the combined constraint set is consistent (including the $\wedge s_i = t_i$ parameter passing constraints). In this case the former node is simply replaced by the root of the other tree. (Some renaming of variables may be necessary to satisfy the requirement that the clause variants that define parent/child relationships in the tree do not share variables with one another.) This observation can be used by a relying party to enable her to make use of IpTs constructed previously and/or by a third party.

Now further observe that any IpT $\mathcal{T}$ can be summarized by a single clause, also called *summary clause*.

Definition 5.2 Let $\mathcal{T} = \langle \mathcal{N}, \mathcal{R}, \mathcal{E} \rangle$ be an identity proof tree. Let $\mathcal{N}' \subset \mathcal{N}$ be the set of open nodes in $\mathcal{T}$. A *summary clause* for $\mathcal{T}$ is a clause of the form $H \leftarrow n_1.\mathit{UpL}_1, \dots, n_k.\mathit{UpL}_k; \phi$, where $H = \mathcal{R}.\mathit{UpL}$, $\mathcal{N}' = \{n_1, \dots, n_k\}$, and ϕ is obtained by performing existential-quantifier elimination on $\overline{\exists}\mathbf{x}.\ \wedge_{n \in \mathcal{T}.N}\ n.\mathit{Const}$, in which $\mathbf{x}$ enumerates the variables in H and in $n_1.\mathit{UpL}_1, \dots, n_k.\mathit{UpL}_k$.

We can now enable the relying party to make use of these summary clauses that are generated by verifiers that the relying party trusts to faithfully construct IpTs and summary clauses. In this way, the relying party can avoid repeating work performed by the trusted verifier. She can also construct valid derivations of authorizations in some cases where her own lack of authorization to receive certain policy rules or credentials would otherwise prevent her doing so. This is because another verifier, authorized to receive the protected policy rules or credentials, can construct an IpT using them, and then make the summarizing clause available to the relying party. In section 6, we study how to ensure that doing this does not disclose to the relying party any information that she is not authorized to receive.

5.3 Timeliness of Identity Proof Trees

As discussed in Section 2, the validity of credentials carrying clauses used to construct an IpT is checked when they are received or when the IpT is constructed. Their validity at the time the IpT is used is not essential. This

[3]The notation $\overline{\exists}\mathbf{x}.\phi$ denotes $\exists\mathbf{y}.\phi$ in which $\mathbf{y}$ comprises all the variables in ϕ not occurring in $\mathbf{x}$. It is used to project constraints on to the variables that appear in a particular formula.

is because the credential is just a secure conveyance of the credential, and the validity of the characteristics inferred by using such clauses is not dependent on the validity of the conveyance. What is essential is that clauses composing the IpT are valid when the IpT is used for authorization purposes.

Identity proof trees carry information on their temporal validity status. This information is of particular importance when a significant amount of time elapses between the validation time and the time the derived characteristic is used for authorization purposes.

The temporal validity of an identity proof tree is influenced by the lifetime of the characteristics that label its nodes. The management of validity time of characteristics is ultimately determined by the semantics of the characteristics themselves. For instance, in order to have a current status as a military veteran, it is necessary only to have been in the military at some point in the past, not to be in the military now. This illustrates that it would be inappropriate to require all characteristics in a clause body to be valid for the characteristic in the clause head to be valid. However some basic patterns for managing characteristic validity time are nicely supported by CLP. In particular, characteristics have a single time parameter that assumes time values during which the characteristic holds, denoted by the variable vt. At the time an authorization decision must be made by determining whether the requester has a given characteristic, the authorization should be granted just in case the characteristic holds at the current time. It is quite straightforward to require that supporting characteristics appearing in the body of some clause used to construct the IpT must be valid for the characteristic in the head of such a clause to be valid.

If the characteristic labeling the root of the tree has time interval parameters, their values are determined by the clauses used for building the tree itself. The semantics of characteristic validity periods is quite variable, and must be specified in the clauses by the policy authors.

Example 5 *Consider the following clause:*

$$\begin{array}{l}\mathrm{DistinguishedAlumni(s, JohnsHopkins,}vt, \mathrm{t}_b, \mathrm{t}_e) \leftarrow \mathrm{StudentId(s, JohnsHopkins,}vt_{old}, \mathrm{t}_{Idb}, \mathrm{t}_{Ide}),\\ \quad \mathrm{Contributor(s, JohnsHopkins,}vt, \mathrm{t}_{\mathrm{issue}}); \mathrm{t}_b < vt, \mathrm{t}_b = t_{issue}, t_{Ide} < vt.\end{array}$$

The clause states that in order to claim the benefits of a DistinguishedAlumni, the claimant should be a former student at JohnsHopkins. The principal's StudentId ending validity should be before the current time (vt), and he should also currently be an active contributor of the school.

Unlike expiration, revocation is unpredictable and can happen at any time. A characteristic may be revoked, invalidating the authorization decision based on the tree in which it appears, even if the characteristic's temporal validity interval has not elapsed. Revocation is handled by the characteristics' issuers, which publish and maintains updated characteristics' revocation lists, listing the serial number of revoked characteristics.

To prevent the relying party from using revoked characteristics, a verifier, while sending the summary clause to the relying party, can send along the serial number of the clauses in the tree that are subject to revocation, without disclosing the actual characteristics. The recipient, once informed of such numbers, can contact appropriate credential-status servers to determine whether the clauses with those numbers have been revoked. We assume that serial numbers are chosen in such a way as to ensure that they do not reveal the clauses they denote.

For example, suppose the characteristics appearing in the body of the clause of Example 5 are verified. The serial numbers of all the clauses used in the derivation are automatically accumulated in the first parameter of the characteristic $\mathrm{Contributor}(\mathrm{diff}(SerialList, []), s, \mathrm{JohnsHopkins}, i, vt_B)$ (which, as discussed above, we have elided in several of the previous examples). When this information is shared with other relying parties, the recipient has the opportunity to utilize revocation publication services or similar means to determine whether any of the clauses used in the derivation have been revoked. If none have, then the derivation remains valid.

6 Safe Disclosure of Characteristics Derived From Sensitive Properties

As discussed in Section 4, we allow principals to author CRP statements that authorize entities with specified characteristics to receive a certain degree of detail about the principal's characteristics, when needed to authorize services. For instance, Alice may allow everyone to know that her salary is between \$15K and \$200K, but be willing to tell her colleagues that the number is between \$40K and \$150K. Moreover, Alice may provide colleagues even greater precision if she is asking them to help finance her travel to an international conference.

Third-party verifiers can assist service requesters in obtaining authorization while protecting their sensitive characteristics. In particular, if a verifier is available that is trusted by the requester with the sensitive information contained in his characteristics, and also trusted by the service provider to faithfully evaluate whether the requester satisfies the appropriate authorization policy, then it may be possible to grant the request in some cases where direct disclosure of the requester's credentials to the service provider would violate the requester's CRP. On the other hand, just the knowledge that the requester is authorized could in some cases disclose to the service provider information about the requester's characteristics that the provider is not authorized to receive. In this section we show how to derive from the requester's CRP additional CRP statements that identify entities that are authorized to know whether a requester is authorized for service, or to know other derived characteristics of the requester.

The determination of whether a particular authorization policy is satisfied can be modeled in terms of the more general problem of determining whether a certain derived characteristic follows from the policy. We assume that there is a distinguished collection of predicates that are defined only by facts, and that only characteristics formed using these predicates can be protected by CRP statements authored by users. Based on these statements, we show how to derive CRP statements that, if satisfied, authorizes a principal to receive characteristics derived from the protected facts. Note that this activity is performed within our system by the verifier whenever it must determine whether a derived characteristic can be released. In particular it is performed by the verifier that determines whether a service requester satisfies the policy authorizing it for the requested service.

The verifier begins the process of determining the authorization policy for the derived fact by constructing an IpT proving that the requester has the derived characteristic of interest. We can imagine this being done in two steps. First the verifier generates a partial IpT $\mathcal{PT}$ in which the open nodes are exactly those nodes labeled by sensitive facts. Second the verifier checks to see whether the sensitive facts can be used to complete the IpT so as to obtain a consistent, closed IpT $\mathcal{CT}$. Of course, these two steps may need to be iterated to search all possible ways of completing such a proof. When and if one is found, we consider the partial IpT $\mathcal{PT}$.

Suppose $\mathcal{PT}$ has a single open node. In this case, $\mathcal{PT}$ can be summarized by a single clause of the form

$$p(\mathbf{s}) \leftarrow q(\mathbf{t}); \phi \tag{1}$$

Further suppose that there is a sensitive fact of the form $q(\mathbf{c})$ that is protected by a CRP statement of the form

disclose $\langle q(\mathbf{t}'); \phi' \rangle$ *if* $\langle \mathbf{R}(\mathbf{u}); \psi \rangle$ *for <purpose-list>*

Here we let $\mathbf{R}(\mathbf{u}); \psi$ denote an arbitrary goal, in which $\mathbf{R}(\mathbf{u}) = p_1(u_1^1, \ldots, u_{k_1}^1), \ldots p_n(u_1^n, \ldots, u_{k_n}^n)$ and there is a single a distinguished variable that occupies the subject field u_1^i of each atom, which represents the subject that is the would-be recipient of the disclosure. All the variables except this one[4] are implicitly existentially quantified, making $\mathbf{R}(\mathbf{u}); \psi$ in effect a predicate on principals that holds for principals that are eligible to receive the information

$$q(\mathbf{t}') \text{ holds for some valuation of the variables it contains that satisfies } \phi' \tag{2}$$

Let us assume that $(\mathbf{t}' = \mathbf{c}) \wedge \phi'$ is satisfiable, which means that the valuation induced by $\mathbf{t}' = \mathbf{c}$ is a witness to (2). Further assume that $\phi' \wedge (\mathbf{t}' = \mathbf{t}) \rightarrow \overline{\exists}\mathbf{x}.\phi$, in which $\mathbf{x}$ comprises the variables appearing in $q(\mathbf{t})$. This

[4]This variable is also not permitted to appear in $q(\mathbf{t}'); \phi'$.

means that (2) is sufficient to satisfy the body of clause 1. Now it follows that, because we assume that the clauses composing $\mathcal{PT}$ are not sensitive, any principal satisfying $\langle \mathbf{R}(\mathbf{u}); \psi \rangle$ is authorized for the information

$$p(\mathbf{s}) \text{ holds for some valuation that satisfies } \phi'' \tag{3}$$

in which $\phi'' = \overline{\exists}\mathbf{y}.(\phi \wedge \phi' \wedge (\mathbf{t}' = \mathbf{t}))$ and $\mathbf{y}$ comprises the variables appearing in $p(\mathbf{s})$. Thus we can use the following CRP to govern disclosure of this information:

disclose $\langle p(\mathbf{s}); \phi'' \rangle$ *if* $\langle \mathbf{R}(\mathbf{u}); \psi \rangle$ *for <purpose-list>*

The techniques discussed above can be extended to handle the general case in which there are multiple CRP statements governing disclosure of characteristic q and there are multiple sensitive facts cause the IpT to have multiple open nodes. This material is presented in the appendix for the interested reviewer. However space constraints prevent its inclusion in the main body of the paper. Should the paper be accepted, the material will be made available through a technical report.

7 Related work

Our research stands at the crossroads of different research areas, such as digital identity management, authorization and trust management systems and declarative logic proof trees. In this section we overview related work from the mentioned areas, and focus on the work closest to ours.

Identity systems are used by on-line service providers to establish and manage users identities, that are expressed in terms of users' attributes. Digital identity management systems typically are defined in terms of one or more service providers cooperating together, thus forming *federations*. Sharing and management of users profiles is demanded to the core entities of identity systems, that is, *identity providers*. Several federation models have been developed, interesting work can be found in [] and in []. Specifically, in the corporate world several emerging standards for identity federation have emerged, like Liberty Alliance [8] (LA) and WS-Federation [20]. The main goal of such initiatives is to define standards for identity verification in a decentralized fashion. Identity is defined in terms of users' attributes which are kept at the identity provider and distributed to service providers on a need to know basis. We see our work complementary to such body of work in that identity federation do not specify how the validity of users' attributes can be verified. Attributes are generally assumed to be true and valid. In this work we rather focus on verification of users' characteristics, so that attribute decisions can be safely taken upon.

In recent years, a number of researchers have developed sophisticated access control models in which access control requirements may be expressed by using rules that are employed to reason about authorized forms of access (see, for example, [1], [2], and [14]). In these approaches, access to resources are expressed by using rules that define the conditions that must be satisfied in order for a permission/ denial/authorization to hold. Barker and Stuckey [2] show how to use constraint logic programming to specify flexible access control policy. The approach described in [2] makes use of specialized constraint solvers, for the efficient evaluation of access requests in situations where large numbers of parametric derivation rules would be expensive to compute, and when changes to an access policy are performed dynamically as a consequence of a users session management. In [14] Li and Mitchell show that by adequately limiting the expressive power of the constraint domains used, DATALOG with constraints can be efficient (polynomial-time evaluation).

Cassandra [1] is a policy language and system for a UK electronic health records system that, as well as our system, uses DATALOG with constraints. Cassandra is role-based; it supports credential-based access control and rules can refer to remote policies. The policy language together with the access control semantics provide a rigorous framework unifying dynamic RBAC, role revocation, distributed trust management and trust negotiation.

Another area that is strongly related to the work presented in this paper is that of trust management. Trust management has been proposed as an approach for protecting open, decentralized systems, in contrast to traditional tools for securing conventional systems [4]. The idea of a trust management system is basically to provide multicentric access control, based on distributed policy statements issued by multiple principals.

At present, the best-known trust-management system is KeyNote [5]. Keynote was designed for use in managing authorizations for small and large scale Internet-based applications. It provides a single, unified language for both local policies and credentials. KeyNote credentials, called "assertions," describe delegations in terms of actions that are relevant to a given application. Thus, KeyNote focuses on delegating authorization to use a specific resource. It is not well suited for asserting more general characteristics of users, nor for basing authorization or trust decisions on such characteristics. Additionally, KeyNote does not support specification of attribute release policies, nor does it provide enforcement mechanisms for preserving confidentiality of attributes.

Another significant related work is represented by Delegation Logic [13]. Delegation logic is a logic-based language to represent policies, credentials, and requests in distributed authorization. Delegation logic extends Datalog with expressive delegation constructs that feature delegation depth and a wide variety of complex principals. It is based on model-theoretic semantics and has further been extended to support non-monotonicity, negation and prioritized conflict handling.

SD3 [9] is an inference engine for a trust management system that constructs a proof tree for a given query so that the querier can verify the correctness of the query result. The policy language in SD3 is an extension of Datalog, and security policies are a set of assumptions and inference rules. However, the specification of constraints is not supported. Additionally, SD3 does not have a mechanism to allow the information in policies to be kept private from certain parties, which we accomplish with specifying the purpose and the constraints in the attribute release policies. The focus of [9] is to retrieve certificates (that correspond to facts in a knowledge base) from remote hosts automatically, and a whole proof tree is constructed on a central server. All the remote hosts must trust the central server to preserve the confidentiality policies of their facts. The technique we provide for proof verification of such certificates is not based on a centralized host, and it can be performed and parallelized among many hosts. Additionally, by using abduction, in our work hosts can verify in a systematic manner the inferred information that a receiver would obtain by releasing information; and thus automatically check that the requirements of the attribute release policies of principal's characteristics' owners are met. This feature is not supported by SD3, in that certificate retrieval is automatic and assumes the certificates to be simply available and not controlled by specific rules.

A growing body of work exploiting the concept of *properties* of the entities as a means for establishing trust is given by *trust negotiation* (TN) [22, 17]. A trust negotiation consists of the iterative disclosure of digital credentials, representing statements certified by given entities for verifying properties of their holders in order to establish mutual trust. Particular focus has been posed on privacy issues related with trust negotiation systems [3, 18, 23], as it has emerged as one of the main obstacles to the wide spread of trust negotiation systems. Elegant solutions have been proposed over the years to achieve various degrees of privacy-preservation through minimal-disclosure of credentials. In addition, some mechanisms and cryptographic techniques have been proposed, dealing with hidden credentials [7, 11] and oblivious credentials. The aim of these techniques is to let a negotiating party fulfill a sensitive policy without need of reading its content and/or showing the actual credential content while satisfying the policy.

Li et al. [12] presented a system for protecting sensitive personal information while using it to gain authorization through the use of automated trust negotiation. Techniques are introduced that integrate various cryptographic protocols that can be used, for instance, to prove that an attribute value lies within a certain range without revealing the exact value. A language for policies is introduced that enables certain forms of partial information to be released based on the recipient's characteristics. By comparison our policy language is more uniform and more flexible[5], permitting as it does the authorized degree of detail to be specified by arbitrary

[5]The exception to our greater flexibility is that we do not support releasing "one bit" of information in the sense of indicating

constraints.

Another important contribution in this area is represented by [6]. Samarati et al. proposed a framework for regulating access and information release on the web. The framework is targeted for decentralized environments. The authors define a policy language for access control specification and a filtering mechanism to identify the relevant policies for a negotiation between a client and a server. A service rule is composed of prerequisite and a requisite rule. To protect both server and client's privacy, the model enforces a specific ordering between a service's prerequisite and a service requisite rule. We greatly differ from Samarati's model, as we define a multi-party model, where multiple service providers are actively involved for verification of identities purposes, and we do not focus on filtering mechanism for access control decisions. We also propose a different approach to deal with privacy and confidentiality of data, which is based on the notion of attribute release policies and on abduction, as discussed in Section 6.

PeerTrust [16] is a management system that uses a simple policy language based on logic programs language. The underlying trust negotiation protocols are similar to the one discussed in [21, 17]. PeerTrust policies are evaluated by a Datalog-based logic engine that allows delegation of authority, singed rules, expression of complex conditions and sensitive policies. The prototype system however, does not support specific confidentiality mechanisms nor deals with timeliness issues.

Proof Trees have been used in previous work for authorization purposes. In particular, our work borrows some ideas from that of Minami and Kotz [15]. The authors propose a secure context-sensitive authorization system that protects confidential information in facts or rules. The system allows multiple hosts in a distributed environment to perform the evaluation of an authorization query in a collaborative way; with no need of a universally trusted central host that maintains all the context information. The proof for making an authorization decision into a set of sub-proofs is produced on multiple different hosts, while preserving the integrity and confidentiality policies of the principals operating these hosts. Our approach follows a similar philosophy, but we differ along several dimensions. First, we base our syntax on constraint Datalog, while the authors in [15] use simple logic rules. The use of constraints in policy statements allows for fine-grained control of structured characteristics, and thus ensure higher expressiveness than simple logic rules. Additionally, constraints provide a method to protect information about attribute values differentially depending on the precision of that information. Second, our work is more flexible in that any IpT can be summarized by a single clause and used by the relying party with no need of additional security policies. Additionally the relying party can also construct valid derivations of authorizations in some cases where his own lack of authorization to receive certain policy rules or credentials would otherwise prevent her doing so.

Finally, an aspect that is not included in Minami's work deals with the timeliness of the validation. By introducing the current validation time in each characteristic -and thus in each node- we enable multiple usage of the same identity proof tree and ensure that authorization decisions are taken only on the basis of temporarily valid facts.

8 Conclusion and Future work

In this paper we provide an approach according to which an entity, referred to as verifier, can satisfy a query proving some property related to the identity of a subject. The process of evaluation of a query against a subject identity is traced through a structure, referred to as identity proof tree, that carries all information proving that a policy requirement is met. Our system is not intended to be a concrete representation for exchange of identity proof trees, and a suitable one is certainly needed. As future work, we plan to elaborate a precise computational model defining how multiple verifiers, services providers, issuers cooperate for information exchange. Subsequently, we will develop an XML-based protocol to support the exchange of proof steps.

whether or not the attribute values satisfy a policy given by the opponent. If this process is iterated, the opponent can learn the sensitive data with arbitrary precision. We do not support this.

References

[1] Cassandra: Distributed access control policies with tunable expressiveness. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)*, pages 159–168, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Steve Barker and Peter J. Stuckey. Flexible access control policy specification with constraint logic programming. *ACM Trans. Inf. Syst. Secur.*, 6(4):501–546, 2003.

[3] Elisa Bertino, Indrakshi Ray, Anna C. Squicciarini, and Elena Ferrari. Anonymity preserving techniques in trust negotiations. In *To appear in proceedings of 5th Privacy Enhancing Technologies Workshop, Dubrovnik, Croatia*, 2005.

[4] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[5] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, page 164, Washington, DC, USA, 1996. IEEE Computer Society.

[6] Piero A. Bonatti and Pierangela Samarati. A uniform framework for regulating service access and information release on the web. *J. Comput. Secur.*, 10(3):241–271, 2002.

[7] Robert W. Bradshaw, Jason E. Holt, and Kent E. Seamons. Concealing complex policies with hidden credentials. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 146–157, New York, NY, USA, 2004. ACM Press.

[8] http://www.projectliberty.org. Liberty Alliance Project.

[9] Trevor Jim. Sd3: A Trust Management System with Certified Evaluation. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 106, Washington, DC, USA, 2001. IEEE Computer Society.

[10] Paris C. Kanellakis, Gabriel M. Kuper, and Peter Z. Revesz. Constraint Query Languages. In *9th ACM Symposium on Principles of Database Systems (PODS)*, pages 299–313, 1990.

[11] Jiangtao Li and Ninghui Li. Oacerts: Oblivious attribute certificates. *IEEE Trans. Dependable Secur. Comput.*, 3(4):340–352, 2006.

[12] Jiangtao Li, Ninghui Li, and William H. Winsborough. Automated Trust Negotiation using cryptographic credentials. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 46–57, New York, NY, USA, 2005. ACM Press.

[13] 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.

[14] Ninghui Li and John C. Mitchell. DATALOG with Constraints: A Foundation for Trust Management Languages. In *Proceedings of the 5th International Symposium on Practical Aspects of Declarative Languages*, pages 58–73, London, UK, jan 2003. Springer-Verlag.

[15] Kazuhiro Minami and David Kotz. Secure Context-Sensitive Authorization. In *PerCom*, pages 257–268, 2005.

[16] Wolfgang Nejdl, Daniel Olmedilla, and Marianne Winslett. PeerTrust: Automated Trust Negotiation for Peers on the semantic web. In *Workshop on Secure Data Management in a Connected World (SDM'04)*, Toronto, Canada, August 2004.

[17] K. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for policy languages for trust negotiation, 2002.

[18] Kent Seamons, Marianne Winslett, Ting Yu, Brian Smith, E. Child, and J. Jacobsen. Protecting privacy during on-line trust negotiation. In *Proceedings of 2 nd Workshop on Privacy Enhancing Technologies, San Francisco, CA, April 2002, to appear.*, 2002.

[19] Leon Sterling and Ehud Shapiro. *The art of Prolog: advanced programming techniques*. MIT Press, Cambridge, MA, USA, 1986.

[20] IBM Microsoft RSA VeriSign. Web Services Federation Language (WS-Federation). version 1.0. July 8 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-federation.asp.

[21] William H. Winsborough and Ninghui Li. Towards practical automated trust negotiation. In *Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, pages 92–103. IEEE Computer Society Press, June 2002.

[22] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*, volume I, pages 88–102. IEEE Press, January 2000.

[23] Ting Yu and Marianne Winslett. A Unified Scheme for Resource Protection in Automated Trust Negotiation. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 110–122, 2003.

Appendix

The following presents how the techniques developed in section 6 can be extended and generalized. First, we consider the case in which there are multiple CRP statements for the atom labeling an open node. Then we show how to handle the general case in which multiple sensitive facts cause the IpT to have multiple open nodes. This material will be included in a technical report and cited in the final paper, should it be accepted for publication.

We now consider the case in which there is just one open node, but there are m CRP statements for q. Assume they are as follows:

$$\begin{array}{c} \textit{disclose}\ \langle q(\mathbf{t}_1); \phi'_1\rangle\ \textit{if}\ \langle \mathbf{R}_1(\mathbf{u}_1); \psi_1\rangle\ \textit{for <purpose-list>} \\ \vdots \\ \textit{disclose}\ \langle q(\mathbf{t}_m); \phi'_m\rangle\ \textit{if}\ \langle \mathbf{R}_m(\mathbf{u}_m); \psi_m\rangle\ \textit{for <purpose-list>} \end{array}$$

In this case, there are two approaches. If adding constraints on the variables of $\mathbf{t}$ in clause 1 has the effect of constraining the values of variables in $\mathbf{s}$ and we want to ensure that the information that can be disclosed under our derived CRP statements is as precise as possible, then we introduce up to m CRP statements of the form

$$\textit{disclose}\ \langle p(\mathbf{s}); \phi''_i\rangle\ \textit{if}\ \langle \mathbf{R}_i(\mathbf{u}_i); \psi_i\rangle\ \textit{for <purpose-list>}$$

in which $1 \leq i \leq m$, $\phi''_i = \overline{\exists}\mathbf{y}.(\phi \wedge \phi'_i \wedge (\mathbf{t}' = \mathbf{t}))$, $\mathbf{y}$ comprises the variables appearing in $p(\mathbf{s})$, there exists a fact $q(\mathbf{c}) \in \mathcal{P}$ such that $(\mathbf{t}_i = \mathbf{c}) \wedge \phi'_i$ is satisfiable, $\phi'_i \wedge (\mathbf{t}_i = \mathbf{t}) \rightarrow \overline{\exists}\mathbf{x}.\phi$, and $\mathbf{x}$ comprises the variables appearing in $q(\mathbf{t})$.

The second approach is appropriate when either the variables in $\mathbf{t}$ are independent of those in $\mathbf{s}$, or it is deemed more important to limit the number of CRP statements generated than it is to ensure that the disclosable information is as precise as possible. In this case we can introduce just one CRP statement of the following form:

$$\textit{disclose}\ \langle p(\mathbf{s}); \phi\rangle\ \textit{if}\ \langle rq(Ids, u, v); \textit{true}\rangle\ \textit{for <purpose-list>}$$

in which rq is a new auxiliary predicate having three parameters, a list of clause serial numbers, the subject, who is the would-be recipient of the derived characteristic, and the issuer, which is the verifier. The predicate is defined by up to m clauses of the form

$$rq(u_1^1, v) \leftarrow \mathbf{R}_i(\mathbf{u}_i); \psi_i$$

in which $1 \leq i \leq m$, $(\mathbf{t}_i = \mathbf{c}) \wedge \phi_i'$ is satisfiable, $\phi_i' \wedge (\mathbf{t}_i = \mathbf{t}) \rightarrow \overline{\exists}\mathbf{x}.\phi$, and $\mathbf{x}$ are the variables appearing in $q(\mathbf{t})$. This second approach becomes of particular importance in the case in which the $\mathcal{PT}$ contains several open nodes, which we consider presently.

Note that if the would-be recipient of the derived fact knew that $q(\mathbf{c})$ can hold for only one tuple of values $\mathbf{c}$, then he would be able to infer from knowing that there exists a valuation satisfying, say, ϕ_1'' that makes $p(\mathbf{s})$ true and that there exists a valuation satisfying, say, ϕ_2'' that makes $p(\mathbf{s})$, then there must exist a valuation satisfying $\phi_1'' \wedge \phi_2''$.

Multiple open nodes We now generalize our analysis further to the case in which multiple sensitive facts cause the IpT to have multiple open nodes. In this case, the summary clause has the following form:

$$p(\mathbf{s}) \leftarrow q_1(\mathbf{t}_1), \ldots, q_\ell(\mathbf{t}_\ell); \phi \tag{4}$$

We assume that the second of the two approaches is taken to dealing with the disclosed constraint ϕ. Assume the ARP statements in question are as follows:

$$\begin{gathered}
\textit{disclose}\ \langle q_1(\mathbf{t}_{1,1}); \phi'_{1,1}\rangle\ \textit{if}\ \langle \mathbf{R}_{1,1}(\mathbf{u}_{1,1}); \psi'_{1,1}\rangle\ \textit{for <purpose-list>}\\
\vdots\\
\textit{disclose}\ \langle q_1(\mathbf{t}_{1,m_1}); \phi'_{1,m_1}\rangle\ \textit{if}\ \langle \mathbf{R}_{1,m_1}(\mathbf{u}_{1,m_1}); \psi'_{1,m_1}\rangle\ \textit{for <purpose-list>}\\
\vdots\\
\textit{disclose}\ \langle q_\ell(\mathbf{t}_{\ell,1}); \phi'_{\ell,1}\rangle\ \textit{if}\ \langle \mathbf{R}_{\ell,1}(\mathbf{u}_{\ell,1}); \psi'_{\ell,1}\rangle\ \textit{for <purpose-list>}\\
\vdots\\
\textit{disclose}\ \langle q_\ell(\mathbf{t}_{\ell,m_\ell}); \phi'_{\ell,m_\ell}\rangle\ \textit{if}\ \langle \mathbf{R}_{\ell,m_\ell}(\mathbf{u}_{\ell,m_\ell}); \psi'_{\ell,m_\ell}\rangle\ \textit{for <purpose-list>}
\end{gathered}$$

In this case we can introduce one CRP statement of the following form:

$$\textit{disclose}\ \langle p(\mathbf{s}); \phi\rangle\ \textit{if}\ \langle rq_1(Ids_1, u, v), \ldots, rq_\ell(Ids_\ell, u, v); \textit{true}\rangle\ \textit{for <purpose-list>}$$

in which for each j, $1 \leq j \leq \ell$, each rq_j is a new auxiliary predicate having three parameters as above and being defined by up to m_j clauses of the form

$$rq_j(u_1^1, v) \leftarrow \mathbf{R}_{j,i}(\mathbf{u}_{j,i}); \psi_{j,i}$$

in which $1 \leq i \leq m_j$, $(\mathbf{t}_{j,i} = \mathbf{c}) \wedge \phi'_{j,i}$ is satisfiable, $\phi'_{j,i} \wedge (\mathbf{t}_{j,i} = \mathbf{t}) \rightarrow \overline{\exists}\mathbf{x}.\phi$, and $\mathbf{x}$ comprises the variables appearing in $q_1(\mathbf{t}_1), \ldots, q_\ell(\mathbf{t}_\ell$.

In this case we can introduce one CRP statement of the following form:

$$\textit{disclose}\ \langle p(\mathbf{s}); \phi\rangle\ \textit{if}\ \langle rq(Ids, u, v); \textit{true}\rangle\ \textit{for <purpose-list>}$$

in which rq is a new auxiliary predicate, having three parameters as above, that is defined by up to $\Pi_{j=1}^{\ell} m_j$ clauses of the form

$$rq(u_1^1, v) \leftarrow \mathbf{R}_{1,i_1}(\mathbf{u}_{1,i_1}), \ldots, \mathbf{R}_{\ell,i_\ell}(\mathbf{u}_{\ell,i_\ell}); \psi_{1,i_1} \wedge \cdots \wedge \psi_{\ell,i_\ell}$$

in which for each $1 \leq j \leq \ell$, $1 \leq i_j \leq m_j$ and there exists a fact $q_j(c_j) \in \mathcal{P}$ such that $(\mathbf{t}_{j,i_j} = \mathbf{c}_j) \wedge \phi'_{j,i_j}$ is satisfiable, and $\bigwedge_{j=1}^{\ell} \phi'_{j,i_j} \wedge (\mathbf{t}_{j,i_j} = \mathbf{t}_j)$ is satisfiable and entails $\overline{\exists}\mathbf{x}.\phi$, in which $\mathbf{x}$ comprises the variables appearing in $q_1(\mathbf{t}_1), \ldots, q_\ell(\mathbf{t}_\ell)$.

Obviously the number of clauses that can be introduced here is combinatorial in the number of sensitive atoms used in the IpT. However, it is not necessary to generate all such clauses. Any clause generated in the manner described above, if it is shows that the recipient is authorized, is sufficient to allow the verifier to provide the derived characteristic in good faith to the requester.