Requirements for Policy Languages for Trust Negotiation

Kent E. Seamons[1], Marianne Winslett[2], Ting Yu[2],
Bryan Smith[1], Evan Child[1], Jared Jacobson[1], Hyrum Mills[1], Lina Yu[1]

[1]*Computer Science Department, Brigham Young University*
Provo, UT, USA 84602
{seamons,bjcmit,childe,millsh,lina}@cs.byu.edu
[2]*Department of Computer Science, University of Illinois at Urbana-Champaign*
Urbana, IL, USA 61801
{winslett,tingyu}@cs.uiuc.edu

Abstract

In open systems like the Internet, traditional approaches to security based on identity do not provide a solution to the problem of establishing trust between strangers, because strangers do not share the same security domain. A new approach to establishing trust between strangers is trust negotiation*, the bilateral exchange of digital credentials describing attributes of the negotiation participants. This approach relies on access control policies that govern access to protected resources by specifying credential combinations that must be submitted to obtain authorization. In this paper we describe a model for trust negotiation, focusing on the central role of policies. We delineate requirements for policy languages and runtime systems for trust negotiation, and evaluate four existing policy languages for trust management with respect to those requirements. We conclude with recommendations for extending existing policy languages or developing new policy languages to make them suitable for use in future trust negotiation systems.*

1. Introduction

In open systems like the Internet, sensitive business transactions frequently occur between strangers, entities that have no pre-existing trust relationship and do not belong to the same security domain. For example, suppose Alice is a student who, while purchasing textbooks over the Internet, discovers an on-line bookstore that offers a student discount. Alice submits her purchase order and checks a box requesting the student discount. At that point, Alice does not know what credentials, if any, the web site might require. Upon receiving her order, the bookstore requests that Alice submit her digital student ID card and a digital credit card. Alice is willing to submit her student ID card, but she has a policy that she will only submit her credit card to members of the Better Business Bureau (BBB). In addition, she is only willing to submit private information to businesses that have a privacy policy that prohibits sharing of private information with any outside organization unless Alice gives explicit permission. Before disclosing her credentials, Alice requests a BBB credential from the bookstore. She also requests that the bookstore disclose a TRUSTe credential certifying that the bookstore adheres to the opt-in privacy handling policy that Alice requires. The bookstore discloses these two credentials to Alice, who then discloses her student ID and credit card so that she can purchase her textbooks with a student discount.

This example is an illustration of *trust negotiation*, a new approach to building trust on-line through the bilateral exchange of digital credentials issued by trusted third parties [15][13][6]. Credentials must be verifiable and unforgeable, so we adopt public key certificates, such as X509v3 certificates, to obtain the necessary guarantees. A credential may contain sensitive information whose disclosure must be carefully managed in accordance with an access control policy (*policy*, for short) that specifies which credentials must be received before it can be disclosed. Such policies can govern access to all sensitive resources, including credentials, roles, capabilities, policies, and services.

The purpose of this paper is to outline the requirements for policy languages for trust negotiation, and use the requirements to evaluate existing policy languages for trust management, with the goal of

influencing policy language designers when they extend existing languages or design new languages. We also point out a number of improvements or extensions to existing languages and runtime systems that would make these systems better suited to trust negotiation.

2. Trust negotiation model

Our model for trust negotiation is peer-to-peer: both parties may possess sensitive services, credentials, and policies. Trust negotiation is triggered when Bob[1], the party acting as the client, makes a request to Alice, the party acting as a server and controlling access to a sensitive resource according to an access control policy. The policy specifies the combination of credentials that Bob must submit in order to gain access to the service. Because a credential can contain sensitive information, its disclosure is governed by an access control policy that specifies the credential combination that Alice must disclose to Bob before Bob will disclose the credential to Alice. Both Alice and Bob possess credentials, allowing mutual trust to be established as necessary. Alice establishes trust in Bob before Alice discloses sensitive credentials, policies, and services. Bob establishes trust in Alice prior to disclosing sensitive credentials and policies.

The runtime system for trust negotiation includes a negotiation manager at each party to control all aspects of the negotiation. The manager adopts a negotiation *strategy* to determine which credentials and policies to disclose, and when to disclose them. The runtime system relies on a *compliance checker* to test whether access control policies are satisfied.

A naive approach to trust negotiation would be for Bob to disclose all his credentials to an unfamiliar server, Alice, whenever Bob makes a request. In the event the service is protected by an access control policy, Alice can check whether or not Bob possesses the requisite credentials. This simple approach is akin to a customer entering a store for the first time, plopping down their wallet or purse on the counter, and inviting the merchant to rifle through its contents to determine whether or not to trust the customer. Obviously, this is an unacceptable solution to the problem of trust establishment that completely ignores credential sensitivity.

A more reasonable approach that considers credential sensitivity is to have each party repeatedly disclose every credential whose access control policy has been satisfied by the credentials disclosed so far by the other party. However, this shotgun approach lacks focus and may result in many unnecessary credential disclosures, as well as needless rounds of negotiation when failure is inevitable. Like the approach that follows, this approach is vulnerable to denial-of-service style attacks where clients waste servers' time with pointless rounds of trust negotiation, and vice versa. We expect negotiation managers to guard against such attacks by employing patience limits---external limits on the time each party is willing to devote to a negotiation for a particular kind of sensitive resource. As these limits will be imposed from outside the negotiation itself, we do not discuss them further here.

A third approach is for each party to disclose its access control policies that are relevant to the current negotiation, to focus the negotiation and base disclosures on a need to know. Access control policy disclosures inform the other negotiation participant of the trust requirements to satisfy in order to advance the state of the negotiation and obtain access to more and more sensitive resources. However, policy disclosure during trust negotiation places new demands on policy compliance checkers. This paper focuses on policy language issues that arise due to access control policy disclosure, a novel facet of the trust negotiation approach.

3. Policy language requirements for trust negotiation

In this section, we outline the requirements for a policy language and the associated runtime compliance checker for use during trust negotiation. First, we present requirements for policy language expressiveness and semantics. Second, we detail requirements for a runtime compliance checker.

3.1. Policy language expressiveness and semantics

We have identified the following requirements for trust negotiation policy language expressiveness and semantics.

Well-defined semantics. A policy language must have a well-defined semantics, so that Bob can have confidence that when he thinks that he has credentials that satisfy Alice's disclosed policy, Alice will also conclude that those credentials satisfy the policy (and vice versa). We consider a policy language's semantics to be well-defined if the meaning of a policy written in that language is independent of the particular implementation of that language. Ideally, a language should have a simple, compact, mathematically defined semantics, such as logic programs and relational algebra do, but Prolog, Fortran, C, and SQL do not.

Monotonicity. A policy language for trust negotiation should be monotonic, in the sense that if two parties successfully negotiate trust, then that same negotiation should also succeed if accompanied by

[1] For simplicity, we will use Bob and Alice as the names of the negotiating parties in all examples in this paper.

additional disclosures. In other words, disclosure of additional credentials and policies should only result in the granting of additional privileges. As an example, suppose a convicted felon were issued a *ConvictedFelon* credential. If a web service should not be made available to convicted felons, imagine a policy language that grants access to the service as long as a *ConvictedFelon* credential is not disclosed. Any client possessing a *ConvictedFelon* credential could gain access to the system by failing to disclose the credential in question.

The monotonicity requirement does not mean that negation must be banished from policy languages for trust negotiation. For example, inequality constraints on credential attribute values do not violate the monotonicity requirement. Further, checks for the absence of a certain credential can be included in policies, as long as the policy language allows the policy writer to make clear that the party that owns the policy is responsible for checking for the absent credentials. For example, it should be the server's responsibility to check for the absence of a *ConvictedFelon* credential at a clearinghouse for information on criminals, and it should be the server's responsibility to check the VISA clearinghouse site to make sure that a VISA card has not been revoked. The monotonicity requirement also does not prevent the use of temporal predicates to, for example, require that access occur only between 8 AM and 5 PM Eastern Standard Time.

Credential combinations. It is highly unlikely that a single certifying authority will be recognized and trusted to certify all relevant attributes of a subject in a single credential. Thus, we expect electronic wallets to contain many credentials for Internet subjects. In some cases, multiple credentials will be required in tandem in order to establish trust. For example, an adoption service may require a birth certificate for each adoptive parent, plus their marriage certificate. Policies that require a combination of credentials to be submitted may promote greater trust, because an attacker may have to compromise several public/private key pairs before being able to successfully forge multiple credentials and gain illicit access. A policy language for trust negotiation must allow policy writers to require submission of combinations of credentials, using conjunction and disjunction (or other syntax with equivalent expressive power).

Constraints on attribute values. In our model for trust negotiation, credentials are structured objects consisting of multiple attributes, represented as name/value pairs. Each credential has an associated type. The credential type will aid in managing a common ontology so that trust negotiation software can reference standard credential schemas to aid in correctly interpreting credential syntax and semantics. A policy language must allow policy writers to constrain submitted credentials to have a certain type, and restrict the values of any other attribute. For example, Alice should be able to require that her clients submit a valid passport, and that the US government issue the passport.

Inter-credential constraints. When two credentials use different public keys, other attributes such as the subject name may be a strong indicator that the two credentials belong to the same subject. For example, a university may be accredited by ABET and may also be a member of the NCAA. If the name of the university is a match or near match in the ABET and NCAA credentials, a recipient of those two chains may be willing to conclude that the credentials convey information about a single university. The language should allow such constraints to be expressed.

Credential chains. A policy language must provide enough expressive power to describe and constrain chains of credentials, where a subject mentioned in one credential is the issuer of the next credential in the chain. For example, a birth certificate credential may be submitted with an accompanying chain of credentials that demonstrates that the county that issued the birth certificate is a genuine county, through a credential issued by its state, stating that it is a county of that state. Rather than listing all 50 states, plus territories and possessions, the policy writer will probably specify that the chain contain a credential issued by the US State Department, attesting that the state really is a state of the US.

Transitive closure. Trust will be transitive under certain circumstances. For example, if two on-line retailers that Alice trusts both vouch for the trustworthiness of a third on-line business, Alice may be willing to conduct business with the unfamiliar retailer. Some form of transitive closure to allow for credential chains of arbitrary length is useful to express such policies. The language should also allow the policy writer to express constraints on the number or type of intermediate links in such chains.

External functions. A standard library of functions will be needed for operations and comparisons on dates, time, money, etc. These functions must have a well-defined semantics, so that both client and server will agree on whether a particular combination of credentials satisfies that part of a policy.

Local credential variables. We expect that there will be standard, off the shelf policies available for common, widely used resources (e.g., VISA cards, passports). So that such policies can automatically be tailored to their owner, they will need to refer to local credentials belonging to their owner. This will allow, for example, the standard policy for giving out personal information to be automatically tailored differently for adults, adolescents, and young children. Note that the parts of a policy that refer to Bob's local credentials, or to other aspects of Bob's local environment, must be handled

specially during disclosure so that Alice does not try to interpret those parts of the policy with respect to her *own* local environment.

Authentication. The usual model for authenticating credential chains in, e.g., SSL, is for Alice to authenticate Bob as the owner of the leaf credentials in the credential chains that Bob submits. For full flexibility, a policy language must allow explicit specification of authentication requirements, either as part of the language itself or as external function calls. For example, Alice should be able to require that Bob be the child mentioned in a birth certificate credential (e.g., if Bob is applying for a passport), or be the parent of the child mentioned in the birth certificate (e.g., if Bob is applying for a passport for a minor), or be the issuer itself (e.g., if the county applies for a state grant that depends on the number of live births in the county), or to omit authentication requirements entirely (e.g., if a school must submit the birth certificates of all its kindergarteners). An authentication requirement means that at runtime, the credential submitter will have to demonstrate knowledge of the private key associated with a public key referred to in the credential.

Who submits the credentials? The policy language should allow a policy writer to say which part of the policy should be satisfied by credentials that the other party submits, and which part must be satisfied by credentials that the local party should search for (e.g., a check for a revoked credit card or a felony record).

Protecting sensitive policies. An access control policy itself may also be sensitive. By looking at the requirements specified in a disclosed policy, an outsider may infer sensitive information about the local party. For example, if a policy protecting Alice's medical records states that a party (Bob) wishing to access the records must show an employee ID certificate issued either by the local hospital or the mental health department, showing that Bob is a staff physician for one of those two organizations, then people may guess with high confidence that Alice has a mental health problem, which may be very sensitive private information. During trust negotiation, the disclosure of such sensitive policies should also be protected. Such protection can be carried out either at the policy language level (i.e., the syntax and semantics of policy languages provide a means for policy writers to protect sensitive policies) or at the system level (i.e., the runtime system dynamically monitors the disclosure of sensitive policies and adds more constraints when appropriate). In a later section, we will look more closely at the first of these two approaches, in which identification and management of sensitive aspects of policies is an internal aspect of policy creation and is the responsibility of the policy writer, with assistance from policy capture tools that may be trained to help identify sensitive corporate or personal information.

3.2. Compliance checker requirements

We have identified the following requirements for access control policy compliance checkers during trust negotiation.

Compliance checker modes. There are two distinct modes of operation required by a compliance checker during trust negotiation. The first mode is the traditional way a compliance checker functions in a trust management environment, as shown in figure 1. Alice's negotiation manager invokes the compliance checker to determine whether to grant Bob access to a sensitive resource. Alice's negotiation manager provides the compliance checker with Alice's access control policy for the resource, the credentials Bob has disclosed, and possibly other information that Alice has made available locally (e.g., time of day, proof of her age, the results of Alice's checks for revocation of the credentials Bob has submitted, etc.). The compliance checker produces a Boolean result indicating whether or not the credentials satisfy the policy. Some compliance checkers, such as PolicyMaker's, return a justification when access is denied [5]. KeyNote supports the specification of a user-defined justification whenever the result is false [3]. During trust negotiation, the compliance checker adopts this first mode of operation to determine whether to grant access to a sensitive service or to disclose a sensitive credential or policy.

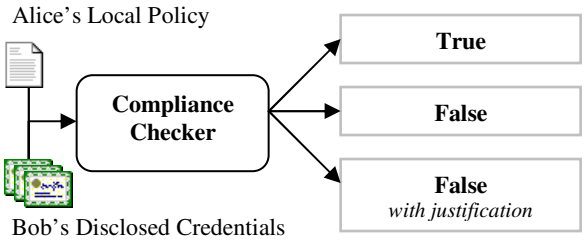

Figure 1: Compliance checker in a traditional mode of operation.

The second mode of operation required of a compliance checker during trust negotiation is not usually necessary in traditional trust management environments, but is required if policies are disclosed during negotiation. Suppose Alice must deny Bob access to a sensitive credential or resource because Bob presents insufficient credentials. Rather than simply deny the request, Alice can disclose the access control policy governing the sensitive resource in order to guide the negotiation to a successful conclusion. Upon receipt of Alice's access control policy, Bob can make use of his compliance checker according to the diagram in figure 2. Bob's negotiation manager invokes the compliance checker to determine whether Bob has credentials cached locally that

satisfy Alice's disclosed policy. The checker accepts Alice's disclosed policy and Bob's local credentials as input, and possibly other information from Bob's environment (time of day, etc.). The compliance checker returns a Boolean result indicating whether Bob has credentials that satisfy Alice's policy. In addition, Bob's negotiation manager needs the compliance checker to return a set of local credentials that satisfies Alice's policy so that those credentials may be disclosed during the negotiation, if Bob's negotiation strategy decides to disclose them. In some cases, Bob may have more than one set of credentials that satisfies Alice's policy. However, some of Bob's credentials in a solution may be sensitive, and Alice may not be able to qualify for access to them. Thus, Bob's compliance checker must be able to eventually find all minimal combinations of credentials that satisfy Alice's policy, in order to exhaust all possibilities for establishing trust. REFEREE is an example of an earlier trust management system that returns a justification when a policy is satisfied.

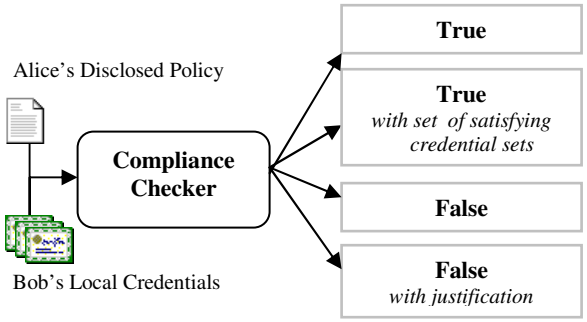

Figure 2: Compliance checker that is able to incrementally determine all minimal sets of credentials that satisfy a remote policy.

Credential validity. During trust negotiation, Bob's runtime system must validate the credentials Alice discloses during trust negotiation, and vice versa. To do this, the integrity of the credential contents must be verified using the digital signature in the credential, to guard against forgery. To accomplish this, the verifier must know or find out the public key of the credential issuer of the root credential in the chain of which this credential is a part. Further, if a policy states that the credential must not be revoked, the runtime system must know how to carry out this check.

Credential ownership. Often Alice's policy will require that Bob be the owner of the credential he submits, the owner of the leaf credential of a credential chain that he submits, or be some other subject mentioned in the credential chain that Bob submits. To support this functionality, Alice's runtime system must challenge Bob to prove ownership of the private key associated with the public key used to identify the subject in the credential. There are various ways this is accomplished in existing security protocols, but the aim is to have Bob sign or decrypt some data with the private key to prove ownership. Care must be taken to prevent an attacker from replaying previously signed data from the legitimate credential submitter.

Credential chain discovery. The supporting credentials needed during a trust negotiation may not be readily available locally. For example, Bob may not have the credential proving that the county that issued his birth certificate really is a legal county. Supporting credentials may need to be gathered for use during a negotiation: this is known as the credential chain discovery problem [11]. A runtime system should include facilities for credential chain discovery; the negotiation strategy should determine when and whether to use those facilities.

4. Evaluation of current policy languages

In this section, we examine four current policy languages for trust management and evaluate them with respect to the requirements presented in the previous section. The languages are PSPL [6], TPL [9], X-Sec [1], and KeyNote [3]. A summary of our analysis of the four languages is shown in figure 3 at the end of the paper.

4.1. PSPL

Bonatti and Samarati [6] present a uniform framework to regulate service access control and information release in an open network. The framework includes a portfolio and service protection language (PSPL), for expressing access control policies for services and release (disclosure) policies for client and server portfolios. The language also includes a policy-filtering mechanism to provide for compact policy disclosures and to protect privacy during policy disclosure. The PSPL language is the only language we are aware of that was designed with many of the needs of trust negotiation in mind. It addresses most, but not all, of the requirements for trust negotiation described in this paper.

A portfolio contains both data declarations and credentials. Unlike credentials, a trusted third party does not certify a data declaration. A client or server submits data declarations and credentials in order to obtain access to sensitive services. The language supports credential-based authorization, departing from the classical assumption that the server has previous knowledge of the requester.

PSPL distinguishes between rules governing services and those governing portfolio data. Rules for services include *prerequisite rules*, *requisite rules* and *facet rules*, while rules for portfolio data include only *requisite rules*. The satisfaction of prerequisite rules is necessary, but not

sufficient, to gain access to a service. *Release rules* govern disclosure of sensitive credentials.

PSPL has a well-defined semantics and is monotonic. It has explicit support for external function invocation, which policy writers can use to express complex constraints on combinations of credentials, including inter-credential constraints and constraints on credential authentication and verification. PSPL does not support inter-credential constraints directly in the language itself.

The only capabilities for trust negotiation that are not represented in PSPL are support for declaring who should submit which credentials contained in a policy, support for transitive closure, and support for credential chain discovery. Although PSPL supports cryptographically signed credentials, there does not appear to be language support for demonstrating credential ownership through possession of a private key.

PSPL is designed to support policy disclosures, to aid in the selection of credentials to disclose to gain access to sensitive resources. Recall that when Alice's policies are evaluated against Bob's local credential store, it is necessary not only to determine if Bob's credentials satisfy Alice's policy, but also (incrementally) to identify all possible combinations that might satisfy Alice's policy, so that all avenues for establishing trust can be efficiently explored. PSPL is the only language we have found that supports both of these modes of operation of Bob's compliance checker.

In PSPL, rules govern access to protected services. A facet of a service can be defined to capture additional or alternative (polymorphic) behavior, i.e., functionalities of the service that can be enabled upon presentation of given credentials or declarations. For example, in an online bookstore, "buy" is regarded as a basic service. When someone wants to buy textbook and receive a student discount, "textbook" is treated as a parameter of the basic service "buy", and "student discount" is treated as a facet of the basic service. Additional requirements are defined to control access to a particular facet of a service.

In PSPL, portfolio disclosure rules govern access to the contents of a portfolio, and service accessibility rules govern access to services. Clients and servers use portfolio disclosure rules to govern access to the contents of their portfolios. Service accessibility rules specify conditions the client must satisfy to gain access to the service. There are two kinds of service accessibility rules, prerequisite rules and requisite rules. Prerequisites are conditions the client must satisfy for a service request to be considered. Requisite rules are conditions that must be satisfied for the service request to be granted.

For example, suppose that Bob will only disclose his VISA card if the server is certified by VISA to collect credit card information. Then the portfolio disclosure rule to govern access to Bob's VISA card credential is as follows:

```
release_reqs (visa_card (issuer = VISA)) ←
credential (certified_server (issuer = VISA,
server = S), K_VISA), current_server (S),
principal(VISA, K_VISA)
```

The `principal` rule defines the association between a principal ***P*** and a public key ***K***. As another example, Bob may have a portfolio disclosure rule specifying that his student ID can be disclosed freely.

```
release_reqs (studentID (issuer = BYU)) ← free
```

Meanwhile, the bookstore has service accessibility rules that govern access to a service to purchase textbooks online. The following requisite rule specifies that a credit card number must be disclosed in order to make a purchase.

```
service_reqs (buy ()) ←
   declaration (credit_card_number = X)
```

In addition, the server also has a facet requisite rule specifying a facet of the service that grants a student discount to students at BYU purchasing textbooks.

```
facet_reqs (buy (material = textbook),
student_discount) ← current_customer (student),
credential (studentID (issuer = BYU, subject =
student), K_BYU), principal (BYU, K_BYU)
```

No prototype implementation of PSPL is publicly available, so we cannot yet comment on the functionality of a compliance checker that supports the language.

4.2. Trust Policy Language

The Trust Policy Language (TPL) [9] developed at IBM Research is an XML-based language that is designed to define the mapping of strangers to roles based on credentials issued by third parties. TPL has a well-defined semantics. There are two types of TPL: Definite TPL (DTPL) and TPL itself. DTPL is a subset of TPL that excludes negative rules, and is, therefore, monotonic.

In TPL, the words *role* and *group* can be used interchangeably. TPL (and DTPL) policies contain a list of definitions of groups using the <GROUP> tag. Inside the group definition are the requirements for membership in that group, in the form of rules using the <RULE> tag. If any of the rules is satisfied, then the requester is a member of the group. As shown in the example below, a RULE entity is composed of a sequence of INCLUSION tags and a FUNCTION tag. An INCLUSION entity indicates basic constraints on common attributes of a credential, such as *type* and *issuer*. More constraints are specified in the FUNCTION entities that follow. Some basic comparison operators and logical connectives between Boolean expressions are supported by TPL syntax. TPL's explicit support for external function invocation enables policy writers to express complex constraints on combinations of credentials, including

inter-credential constraints and constraints on authentication and credential verification.

```
<GROUP NAME="ValidStudent">
<!-- student identification from a university --
>
<RULE>
  <INCLUSION FROM="University" ID="stdntcrt"
TYPE="UnivID">
  </INCLUSION>
  <FUNCTION>
    <EQ>
      <FIELD ID="stdntcrt" NAME="Status"></FIELD>
      <CONST>"student"</CONST>
    </EQ>
  </FUNCTION>
</RULE>
</GROUP>
```

An especially elegant feature of TPL is its support for transitive closure and credential chain discovery. In the Trust Establishment (TE) system [9], mandatory attributes of credentials used with TPL are defined, including *type*, *issuer*, *owner*, *profileURL* and *issuerCertRepository*. The *profileURL* attribute indicates the place where more information about a certain credential type can be found. The *issuerCertRepository* attribute shows the place where more credentials describing the issuer can be found. It is the client's responsibility to submit a set of credentials that satisfies one of the rules. After receiving those credentials, the server needs to check whether the issuers of those credentials are qualified to be members of the groups specified in the INCLUSION entities. At this point, the credential collector of the server starts to probe the issuer's credential repository (indicated by the issuerCertRepository attribute of the client's credentials). Therefore, another role-mapping process is initiated. Such a role-mapping process continues until the server reaches an issuer who is certified by the server, i.e., the server trusts the issuer without any further requirements. Thus, the support for transitive closure and credential chain discovery are combined together in a recursive role-mapping process.

TPL's recursive approach to establishing group membership provides an easy and elegant basis for transitive closure and credential chain discovery. The TE system considers roles to be the only resources that need to be protected. However, any role-based access control system can take over once the client has been mapped to a role or roles, to provide access to a sensitive resource.

The TE system does not provide support for sensitive credentials. One of TE's basic assumptions is that credentials can be disclosed whenever they are requested. It is also assumed that an issuer will put all its credentials in a repository where everybody can access it. Such assumptions are no longer true in automated trust negotiation. Further, the TE system does not have the notion of sensitive policies. Protections for sensitive policies are provided neither in TPL nor in the architecture of the TE system. However, TPL is a good base for extension into a policy language for automated trust negotiation, as these shortcomings can be remedied by extensions to the language and the runtime system.

An example will help to clarify this claim. Recall that in TPL, simple operators, such as value comparison and logical connectives between Boolean expressions, are provided to express the required constraints on credentials. More complex computation is achieved by invoking standard external libraries that implement well-known and frequently used functions. In our view, one such standard function should be the *requester-authenticates-to* function. Its input is a principal's public key. When this function is invoked, a challenge-response process is used to prove that the other party owns the corresponding private key. Consider the following policy summary:

(x.type="birth certificate") AND
(requester-authenticates-to(x.mother) OR
requester-authenticates-to(x.father)) AND
after(x.date_of_birth, 1/1/1983) AND
(x.issuer is from group "government")

The corresponding policy written in DTPL is as follows:

```
<GROUP NAME="1234">
 <RULE>
  <INCLUSION ID="X" TYPE="birth certificate"
                    FROM="government"/>
  <FUNCTION>
   <AND>
    <OR>
    <EXTERN CLASS="RequesterAuthenticatesTo">
      <PARAM NAME="param1">
         <FIELD ID="X" NAME="mother"/>
      </PARAM>
    </EXTERN>
    <EXTERN CLASS="RequesterAuthenticatesTo">
      <PARAM NAME="param1">
         <FIELD ID="X" NAME="father"/>
      </PARAM>
    </EXTERN>
    </OR>
    <EXTERN CLASS="DateAfter">
      <PARAM NAME="param1">
         <FIELD ID="X" NAME="date_of_birth"/>
      </PARAM>
      <PARAM NAME="param1">
         <CONST>1/1/1983<CONST/>
      </PARAM>
    </EXTERN>
   </AND>
  </FUNCTION>
 </RULE>
</GROUP>
```

In this example, a sequence of INCLUSION entities is used to specify the list of potentially relevant credentials, followed by a FUNCTION entity that

represents the function that the required credentials must satisfy. We invoke external functions three times to accomplish complex predicates. In order to create an external function, a class must be created that implements the *TPExternalFunction* interface, which defines the setting, checking, and evaluation of parameters. This class name is specified in the CLASS attribute of the EXTERN element. The first two functions in the example refer to a class called *RequesterAuthenticatesTo* and the third refers to the class *DateAfter*. Within each definition of an external function in TPL is a list of all the input parameters for that function. The PARAM entity sets the name for a function parameter and the FIELD entity sets the value for a function parameter by using the value of an attribute (indicated by the NAME attribute) of a credential (indicated by the ID attribute). The value of an ID attribute should appear in one of the INCLUSION entities.

In most cases, when Bob sends Alice a credential, Alice needs to authenticate Bob to be the owner of that credential. However, the above policy states that a party needs to show his/her child's birth certificate and the child's age must be less than 18. In this case, we need to authenticate Bob to be one of the child's parents rather than the child.

The above example also shows the benefit of including GROUP entities as part of TPL. The policy requires that a third party that can be authenticated to the group *government* be the issuer of the birth certificate. After getting the issuer's public key, Alice's policy compliance checker will verify whether the issuer is qualified to be in the *government* group (using a group definition not shown here). A credential collector will be invoked if necessary, to discover additional relevant credentials. (Note that this offers the possibility of denial-of-service attacks, hence deserves additional attention from researchers in the future.)

To support sensitive policies in automated trust negotiation, a resource's access control policy can be modeled as an interconnected set of policies instead of a single policy. For example, at a web site for a secret cooperative venture of IBM and Microsoft, the server can first ask for an employee credential. If the employee does work for IBM or Microsoft, further requirements can be exposed. Otherwise, the negotiation can fail without disclosing sensitive information.

One possible way to protect sensitive policies is through the use of policy graphs [13]. Formally, a policy graph is an acyclic directed graph with a single source and a single sink. Every node except the sink node represents a policy. The sink represents the protected resource. A policy node *n* can be disclosed to the other negotiation participant only if there is a path from the source to *n*, such that every policy node (except possibly *n*) in the path has been satisfied by credentials disclosed by the other party. The following syntax can be added to TPL to represent a resource's policy graph.

```
<RESOURCE NAME="R004">
     <GROUP NAME="P032">
     ...
     </GROUP>
   .
   .
   .
     <GROUP NAME="P008">
     ...
     </GROUP>
     <EDGE SOURCE="P032" TARGET="P045">
     <EDGE SOURCE="P032" TARGET="P004">
   .
   .
   .
     <EDGE SOURCE="P004" TARGET="R008">
</RESOURCE>
```

The syntax we introduce here for policy graphs is very close to GraphXML [7], which is an XML-based graph description language. The RESOURCE entity has an ID attribute. Inside a resource entity, a sequence of GROUP entities is defined, which correspond to policy nodes in a policy graph. Next a sequence of EDGE entities indicates the relation between those policy nodes. The SOURCE and TARGET attributes show the direction of the edge. Their values must be the names of policies defined earlier, or the name of the resource. Note that the syntax of policy graphs cannot guarantee the graph to be an acyclic graph with a single source and a single sink. It is the policy compliance checker's responsibility to check a policy graph's validity.

This representation of sensitive policies can be used for other policy languages as well, either as a direct addition to XML-based languages or as an XML wrapper around other languages. However, the semantics of the policy language must be extended to apply to policy graphs. Further, the runtime system must be aware that many layers of policies may lie between a party and the resource it would like to access.

4.3. X-Sec

X-Sec is an XML-based language for specifying credentials and security policies for Web document protection [1]. X-Sec was not designed for establishing trust between strangers. Instead, it was designed for use in a system where subjects first subscribe to a document source of interest. The document source defines and stores subject credentials in order to provide credential-based access control to web documents.

In X-Sec, credentials are stored in *subject profiles* and contain attribute values of the credential owner. The following is an example of a faculty credential stored in a profile.

```
<X-profile sbjID="747" PIssuer = "BYU">
   <faculty credID="911", CIssuer = "BYU">
      <name>
         <fname>Kent</fname>
         <lname>Seamons </lname>
      </name>
      <email> seamons@cs.byu.edu </email >
      <position>Assistant Professor</position>
   </faculty>
</X-profile>
```

In X-Sec, access control policies are stored in *policy bases*. The following policy base has a single policy specification, granting access for the Computer Science faculty to view portions of the `students.xml` document.

```
<policy base>
<policy_spec
  cred_expr="//faculty [dept="Computer Science"]"
  target="students.xml", path="//ComputerScience"
  priv="VIEW"  type="GRANT"   prop="CASCADE"/>
</policy base>
```

A policy specification consists of five parts: a `cred_expr` that is an Xpath expression over a subject profile, a `target` denoting the document to which the policy applies, a `path` denoting selected portions within the target document, a `priv` that specifies the security policy access mode (`view`, `navigate`, `append`, `write`, `all`), a `type` specifying whether the security policy is positive (`grant`) or negative (`deny`), and a `prop` to specify the propagation option (`no_prop`, `first_lev`, `cascade`).

X-Sec has a well-defined semantics. X-Sec is not monotonic: it has both *positive* policies that grant access and *negative* policies that deny access. The presence of a credential in a profile that satisfies an Xpath expression in the policy base will result in a denial of access when `type=deny`. Support for negative policies is acceptable in an environment where the server has access to all of the subject's credentials. A restricted form of X-Sec that prohibits negative policies is necessary to make it suitable for use in trust negotiation.

X-Sec has support for specifying credential combinations, constraints on the attribute values contained in credentials, inter-credential constraints, and external functions, all through the use of Xpath expressions. X-Sec has no facilities for local credential variables. The current model does not distinguish between local and remote credentials, since credentials are stored and managed at the service.

There are a number of missing features that would make X-Sec better suited as a language for trust negotiation. X-Sec has no support for credential chains or transitive closure. X-Sec currently has no need for a mechanism to specify chains of trust, since it assumes the client and server are already familiar with one another. Since X-Sec was not designed to have the client submit credentials, X-Sec has no support for specifying the credential submitter or how to authenticate the credential submitter. X-Sec assumes the subject is authenticated through some other means.

There is no publicly available compliance checker supporting X-Sec. An implementation of X-Sec is in progress within the framework of the Author-X project [2]. The design of X-Sec does not require that the compliance checker verify credential validity or prove credential ownership. There is also no need for credential chain discovery, or the generation of all minimal sets of credentials that satisfy an expression.

X-Sec could be extended to support many of the features needed in trust negotiation. The designers have already identified negotiation as an area for future work [1]. Since X-Sec credentials and policies are represented as XML documents, they could be protected using the same mechanisms developed for the protection of Web documents. Incorporating digital signatures into X-Sec credentials will allow verification of subject credentials against forgery. Including the public key of the credential owner in the credential will enable proof of credential ownership through some form of a challenge for possession of the associated private key.

Another possible direction of extension for X-Sec, beyond incorporating support for trust negotiation directly in X-Sec, is for X-Sec to use trust negotiation to automate the subscription process.

4.4. KeyNote

No discussion of languages for trust negotiation would be complete without describing how KeyNote [3], the best-known language for trust management, fits into our framework. The contrast between KeyNote and the other languages we have considered in this section is perhaps most easily seen by considering two small examples. In KeyNote, it is very easy to issue a credential that authorizes, for example, a professor to make purchasing decisions of less than $500 on behalf of BYU. In the other languages we have considered, BYU could issue such a credential, but without standard external schemas and semantics for how vendors interpret such credentials, it is doubtful that most bookstores would understand and accept that credential. On the other hand, in KeyNote it is not natural to issue a credential that says that a certain person is a professor in the Computer Science Department at BYU, and has been employed there since 1992. If there is no notion of delegation of authority inherent in the credential, then only in the simplest of examples will KeyNote's version of the credential be useful for trust negotiation. Since the contents of electronic wallets are, in main part, not intended as delegations of authority, KeyNote does not work well for representing and using such credentials. This is reflected in the fact that KeyNote's credentials do

not consist of attribute name/value pairs, the central feature of the other languages we have considered.

Perhaps the best way to combine the strong points of KeyNote and the other languages we have considered is to embody KeyNote's features in a special class of credentials devoted to declarations of delegation. The attribute values used in these delegation credentials can take on the features present in KeyNote credentials, such as the conditions of use for the credentials. If standard interpreters are made available for the sublanguage used in these delegation credentials, then they can be widely recognized and accepted.

In the remainder of the section, we look at some examples to drive home these points. A KeyNote policy, or *assertion*, is shown below. It consists of a set of fields that describe the principal being authorized (the Licensee), the authorizer of the principal (Authorizer), and a predicate that tests local environment variables, known as the action attribute set in KeyNote. For example, the policy below shows that BYU authorizes Dr. Seamons to perform any action in which the action attribute set satisfies the given condition expression. The principals can be public keys rather than the human readable names shown below, allowing the use of public key authentication procedures. Adding a signature to the policy below makes it a signed policy, or credential, in KeyNote.

```
Authorizer: "BYU"
Licensees:  "Dr. Seamons"
Conditions:  position == "faculty"
```

To illustrate how KeyNote credentials and policies can be adapted for use in a simple trust negotiation scenario, assume an on-line bookstore offers a discount service to faculty members at BYU. The bookstore can establish the following KeyNote policies to specify that faculty members can obtain a discount. The first policy indicates that the bookstore is trusted; the use of the POLICY keyword as the authorizer signifies a trusted root.

```
Authorizer: "POLICY"
Licensees: "UniversityBookstore"

Authorizer: "UniversityBookstore"
Licensees: "BYU"
Conditions:
   if (request == "discount") &&
      (position == "faculty")  -> true
```

When a faculty member requests the discount service and submits their faculty credential, all three assertions are passed to the KeyNote compliance checker, along with an action attribute set comprised of `request=discount && position=faculty`, which evaluates to true.

Extending the faculty discount example shown above, suppose a bookstore offers discounts to faculty members and students at any accredited university. The bookstore creates a policy that delegates authority to ABET to determine an accredited university.

```
Authorizer: "ABET"
Licensees: "BYU"
Conditions:
   accredited == "true"

Authorizer: "UniversityBookstore"
Licensees: "ABET"
Conditions:
   if (request == "discount") &&
       accredited == "true" &&
     ((position == "faculty")  ||
      (position == "student"))   -> true
```

This approach allows the bookstore to accept credentials from faculty members at any accredited university, without having to list all accredited universities in the policy. Accredited universities receive credentials from ABET. When a new client requests a faculty discount or a student discount, the service sets the action attributes appropriately and invokes its compliance checker, passing in the local bookstore policies governing the service and credentials submitted by the client.

This bookstore-purchasing example works well in KeyNote. However, it is not possible to extend the original faculty credential to be a full employee ID. If the credential mentions the professor's department, or any other extraneous information, it will be interpreted as a required or alternative condition for use of the credential---e.g. that the professor is authorized to act on behalf of the computer science department, or that the professor is only authorized to act in situations where the local department is Computer Science.

How does KeyNote fare with respect to the other requirements for trust negotiation policy languages contained in the previous section? KeyNote itself has a well-defined semantics. KeyNote is monotonic, that is, removing an assertion will never result in an increase of privileges. KeyNote supports submission of certain combinations of credentials, through special features in the Licensee field. For example, one can write a policy that says that two vice presidents must authorize all purchases over $10,000.

KeyNote also supports credential chains of arbitrary length, as demonstrated in the example of the online bookstore offering discounts to faculty members at all accredited universities. The Bookstore policy delegates authorization to ABET, who could delegate authorization to accredited universities, who in turn delegate authorization to faculty members. Assuming a standard naming convention in assertions, a bookstore could accept any valid credential chain leading back to ABET. KeyNote also allows control over the length of the chain, by saying, for example, that a certain subject must be the current requestor.

KeyNote has no external functions, though values may be passed into the KeyNote compliance checker from outside. KeyNote does not allow a policy writer to specify which parts of Alice's policy should be satisfied by credentials Bob submits, and which by credentials from Alice's local environment. KeyNote does not provide support for sensitive policies.

The KeyNote compliance checker outputs a user-defined compliance value, which can be as simple as `true` or `false`, or as complex as `No_Access`, `Guest_Access`, `Employee_Access`, or `Admin_Access`. KeyNote's compliance checker does not include functionality to find all the minimal sets of credentials that satisfy a policy. When the compliance checker is invoked, an environment variable is set to the principal the requestor must authenticate to. Actions may be requested by several principals, each considered to have individually requested the action. This provides support for policies requiring multiple authorizations. The KeyNote system has an API for verification of the signatures on policies and credentials, but it does not provide facilities for verifying credential ownership or for gathering credential chains. The system could be enhanced to provide all these functions. KeyNote is designed to provide a simple core set of features, with additional features added as needed at higher levels of the system.

5. Related work

The logic-programming paradigm is a potential basis for the development of suitable trust negotiation policy languages. An early implementation of trust establishment between strangers used Prolog as the policy language [14]. The system design supported portable policies that could be disclosed during trust negotiation and evaluated by a local Prolog interpreter. The system included support for credential chains, credential combinations, and a compliance checker that determined all minimal sets of credentials satisfying a policy.

The development of role-based trust-management languages RT_0 and RT_1 is in progress [11], a follow-on to Delegation Logic [12]. RT_0 supports credential graphs and an approach to the credential chain discovery problem that allows credentials to be stored with subjects or issuers, and for credential gathering to take place in a top down or bottom up fashion. We will explore the RT languages' suitability for trust negotiation once they become publicly available.

6. Conclusions and future work

In this paper, we delineated requirements for policy languages and runtime systems for trust negotiation, and evaluated four existing policy languages for trust management with respect to those requirements.

No current policy language meets all the requirements for trust negotiation. The PSPL language design is the most complete trust negotiation language available today. However, a runtime system is still under development. No compliance checker is available for the X-Sec language either. Once the compliance checkers become available, we intend to conduct experiments using these languages as part of our ongoing TrustBuilder project in automated trust negotiation.

We do not expect a single language to meet all the needs of trust negotiation. An important area to explore in the future is support for parties that adopt different policy languages and wish to negotiate trust with each other.

The existing trust negotiation prototypes that we are aware of [10] use DTPL for trust negotiation, which means that they need extensions to DTPL's compliance checker and the development of additional libraries to provide the missing features that are needed in trust negotiation. For example, the DTPL compliance checker was extended at our request to return a set of credentials that satisfies a policy. Within the TrustBuilder project, we will continue to use DTPL and will also experiment with other alternative runtime systems as they emerge.

We found the most well known trust management language, KeyNote, to be poorly suited in some ways for use in trust negotiation, due to its intended use for delegation of authority and the fact that trust negotiation uses attributes of the negotiation participants as the basis for trust. We believe that a language like KeyNote has a place in trust negotiation as a sub-language for use in specifying constraints on delegation of authority. Further, trust negotiation can be used to automate the acquisition of KeyNote credentials, or capabilities, that can serve as tickets for efficient repeated access to a resource.

In closing, we believe trust negotiation to be an important emerging research area. The design criteria in this paper can serve to guide new language designers on the important features for a future trust negotiation language. The criteria can also guide designers as they extend existing languages to support trust negotiation. The feasibility of such extensions is evidenced by our proposed extensions to DPTL to support policy graphs, sensitive credentials, policy disclosure, and proof of credential ownership.

Acknowledgements

This research was supported by DARPA through AFRL contract number F33615-01-C-0336 and through Space and Naval Warfare Systems Center San Diego grant number N66001-01-18908. We also thank the referees for their helpful comments, and Michael

Halcrow, Adam Hess, and Ryan Jarvis for their help in preparing the final version of this paper.

References

[1] E. Bertino, S. Castano, and E. Ferrari, "On Specifying Security Policies for Web Documents with an XML-based Language," *Sixth ACM SACMAT*, Chantilly, Virginia, May 2001.

[2] E. Bertino, S. Castano, and E. Ferrari, "Securing XML Documents: the Author-X Project Demonstration," *ACM SIGMOD Conference*, Santa Barbara, CA, May 2001.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust-Management System," *RFC 2704*, September 1999.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The Role of Trust Management in Distributed System Security," Chapter in *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, Vitek and Jensen, eds., Springer-Verlag, 1999.

[5] M. Blaze, J. Feigenbaum, and M. Strauss, "Compliance Checking in the PolicyMaker Trust Management System," *Financial Cryptography* 1998: 254-274.

[6] P. Bonatti and P. Samarati, "Regulating Service Access and Information Release on the Web," *7th ACM Conference on Computer and Communications Security*, Athens, Greece, November 2000.

[7] Y. Chu, J. Feigenbaum, B. A. LaMacchia, P. Resnick, M. Strauss, "REFEREE: Trust Management for Web Applications," WWW6 / Computer Networks 29(8-13): 953-964 (1997).

[8] GraphXML. Found on-line at http://www.cwi.nl/InfoVisu/GraphXML/.

[9] A. Herzberg, Mihaeli, Y. Mass, D. Naor, and Y. Ravid, "Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers," *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.

[10] A. Hess, J. Jacobson, H. Mills, R. Wamsley, K. E. Seamons, and B. Smith, "Advanced Client/Server Authentication in TLS," *Network and Distributed System Security Symposium*, San Diego, CA, February 2002.

[11] N. Li, W. Winsborough, and J. Mitchell, "Distributed Credential Chain Discovery in Trust Management (Extended Abstract)," *8th ACM Conference on Computer and Communications Security,* Philadelphia, Nov. 2001.

[12] N. Li, B. N. Grosof, and J. Feigenbaum, "A Practically Implementable and Tractable Delegation Logic," *IEEE Symposium on Security and Privacy*, Oakland, May 2000.

[13] K. E. Seamons, M. Winslett, and T. Yu, "Limiting the Disclosure of Access Control Policies During Automated Trust Negotiation," *Network and Distributed System Security Symposium*, San Diego, CA, February 2001.

[14] K. E. Seamons, W. Winsborough, and M. Winslett. "Internet Credential Acceptance Policies," *Workshop on Logic Programming for Internet Applications*, Leuven, Belgium, July 1997.

[15] T. Yu, M. Winslett, and K. E. Seamons, "Interoperable Strategies in Automated Trust Negotiation," *8th ACM Conference on Computer and Communications Security*, Philadelphia, Pennsylvania, November 2001.

Requirements	PSPL	TPL	X-Sec	KeyNote
Well-defined semantics	Y	Y	Y	Y
Monotonicity	Y	Y (DTPL)	E	Y
Credential combinations	Y	Y	Y	Y
Constraints on attribute values	Y	Y	Y	N
Inter-credential constraints	Y	Y	Y	N
Credential chains	Y	Y	N	Y
Transitive closure	P	Y	N	P
External functions	Y	Y	Y	N
Local credential variables	Y	N	N	N
Authentication	Y	E	N	P
Who submits?	N	N	N	N
Sensitive policies	Y	N	N	N
Compliance checker modes	Y	P	N	N
Credential validity	Y	Y	N	Y
Credential ownership	N	N	N	N
Credential chain discovery	N	Y	N	N

Figure 3. A comparison of four languages for trust management with respect to the requirements for a policy language for trust negotiation (Key: Y-Yes, N-No, E-Easily extended, P-Partial support).