

AFRL-IF-RS-TR-2002-289 Vol 2 (of 2)
Final Technical Report
October 2002



ADVANCED SECURITY PROXY TECHNOLOGY FOR HIGH CONFIDENCE NETWORKS: ADVANCES IN TRUST NEGOTIATION

Trusted Information Systems Network Associates, Incorporated

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J905

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

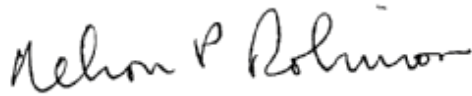
The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-289 Vol 2 (of 2) has been reviewed and is approved for publication.

APPROVED:



NELSON P. ROBINSON
Project Engineer

FOR THE DIRECTOR:



WARREN H. DEBANY, Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> OMB No. 074-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE October 2002	3. REPORT TYPE AND DATES COVERED Final Aug 00 – Jun 01	
4. TITLE AND SUBTITLE ADVANCED SECURITY PROXY TECHNOLOGY FOR HIGH CONFIDENCE NETWORKS: ADVANCES IN TRUST NEGOTIATION			5. FUNDING NUMBERS C - F30602-97-C-0336 PE - 33401G PR - NSA1 TA - 00 WU - 01	
6. AUTHOR(S) William Winsborough				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Trusted Information Systems Network Associates, Incorporated 3060 Washington Rd. Glenwood, MD 21738			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency AFRL/IFGB 3701 North Fairfax Drive Arlington Virginia 22203-1714			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2002-289 Vol 2 (of 2)	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Nelson P. Robinson/IFGB/(315) 330-4110/ Nelson.Robinson@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) This effort addresses the recent trend to fight battles with a coalition of forces from many different commands and countries. This also applies to the business world where companies are cooperating more in order to get new, innovative products on the marketplace, and insure that they all interface together. In order to exchange information between coalitions of organizations (businesses) that have no shared infrastructure and only limited mutual trust, there must be established some sort of bilateral credential exchange, which is called trust negotiation (TN). To accomplish this, a common thread is established throughout this effort to support creation and management of sensitive credentials and policy content to use in attribute-based access control (ABAC). Current ABAC technology is either not sufficiently scalable to meet the needs of dynamic coalitions, or else provides the same access rights to all users. This is usually not desirable when fighting battles or making deals in the business world. In this study, steps are taken toward the goal of making ABAC systems that are highly scalable and fine grained, and to identify issues in the areas of distributed credential discovery, policy language design, and realistic TN strategies.				
14. SUBJECT TERMS Trust Negotiation, TN, Attribute-Based Access Control, ABAC, Trust Management, Credentials, Credential Chain, Credential Discovery, Policy Language, Interoperability, Digital Signature, Identification Card, ID Delegation Logic, DL, Trustbuilder			15. NUMBER OF PAGES 130	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Table of Contents

1. PROJECT OVERVIEW.....	1
1.1. Project Organization.....	2
1.2. Report Organization	3
1.2.1. List of Appendix's.....	4
2. RESEARCH PERFORMED AT NAI LABS	5
2.1. Distributed Credential Chain Discovery.....	6
2.1.1. Lessons Learned and Future Work in Distributed Credential Discovery	8
2.2. Trust Negotiation: Analyzing Language Requirements and Information Flow	8
2.2.1. Lessons Learned and Future Work in Information Flow	12
2.3. Conclusions.....	12
Appendix A Distributed Credential Chan Discovery in Trust Management (Extended Abstract)...	14
Appendix B Making Trust Negotiation Realistic: A Suitable Policy Language and the Management of Information about Credential Possession	24
Appendix C Advanced in Trust Negotiation.....	46
Appendix D Advanced in Trust Negotiation.....	54
Appendix E TrustBuilder: Middleware to Enable Trust Negotiation Strategy Interopreability	57
Appendix F Interoperable Strategies in Automated Trust Negotiation	79
Appendix G Extended version of: Interoperable Strategies in Automated Trust Negotiation	99

1. PROJECT OVERVIEW

The common thread throughout this project is support for creation and management of sensitive credentials and policy content for use in attribute-based access control (ABAC). ABAC seeks to overcome problems of flexibility and scalability in access control systems for dynamic coalitions, such as multilateral military and humanitarian operations, as well as business partnerships and consortia. When resources are shared today, unfortunately many coalitions find themselves with no better alternative than to establish a virtual private network (VPN) and make shared resources available to one another through the VPN. This means that users that have access to any of the shared resources have access to all of them, which is inappropriately coarse granularity of access control. However, existing alternatives, which are based on the identity or capabilities of the resource requestor (including such generalizations as role-based access control) require foreign requestors to be known to the resource-providing organization before access can be authorized. The concomitant administrative overhead makes these systems non-scalable. (Imagine having to keep track of staffing changes in every foreign organization, or even having to issue an explicit capability for each foreign resource each new hire or promotion might require access to.)

Authority in coalitions is inherently distributed. ABAC provides a means for each locus of authority to determine and to specify its own judgements, and for those judgements to be combined naturally to arrive at appropriate authorization decisions. The approach is based on the use of digitally signed attribute credentials through which credential issuers assert their judgements about the attributes of entities, such as users and organizations. Because these credentials are digitally signed, they can serve to introduce strangers to one another. Moreover, the issuers of credentials can be strangers whose authority is determined based on their own attributes, as documented in further credentials. This is one of the keys to ABAC's scalability. A resource requester can be introduced to an access mediator by a chain of credentials. Such a chain starts with a credential issued by an authority known to and trusted by the access mediator, and ends with a credential issued to the requestor. Each credential in the chain is issued by an appropriate authority (for instance, the manager responsible for a staffing change decision), and the authority of that issuer over the judgement in question is in turn based on her attributes (for instance, the manager in question heads up the purchasing department).

Clearly, one of the key issues that ABAC must address is the choice of an appropriate language design. The language is at the core of an ABAC system. It determines the kinds of judgements that can be issued in credentials (e.g., "purchasing agents have spending limits of \$5,000" or "Alice is a purchasing agent"). Furthermore, its semantics determines how the judgements contained in credentials issued by distributed authorities combine to decide authorization questions.

Another key issue is that the data contained in credentials is often sensitive and must be protected. This is central, since it means that the credentials that must be presented to obtain access are themselves subject to access control. Because we are interested in supporting coalitions of organizations that have only limited mutual trust, we believe that the requester and the access mediator will often be unable to agree upon a trusted third-party authorization service. Indeed, it seems debatable whether most individuals—much less pairs of individuals—would be willing to trust any one entity with all of their credentials.

In light of the fact that introducing third parties seems to make the problem harder rather than simpler, the approach we take to managing credential sensitivity calls for requestor and access mediator to enter into a kind of bilateral credential exchange, which we call a trust negotiation (TN). Prior to entering into TN, both entities associate ABAC policies with each of their sensitive credentials. Then when a resource is requested, a TN is conducted to determine whether the access mediator is able to establish sufficient trust in the requestor to grant access. (The requestor may also need to establish trust in the access mediator—a point that we ignore for the moment and return to in the next section.) The negotiation consists of a sequence of credential exchanges that begins by disclosing credentials that are not sensitive. As credentials flow, more are unlocked, until either the policy of the desired resource is satisfied, or the negotiation fails. Many negotiation strategies are possible. Most exchange some form of credential requests, which are derived from the policies governing either other requested credentials or the target resource. Such requests serve to focus the exchange on credentials that are relevant to unlocking the target resource. However, transmitting credential requests raises other issues. For instance, doing so can inadvertently and without authorization provide information about which credentials the transmitter possesses. On-going strategy design work seeks to identify and avoid these potential pitfalls. In addition to needing high quality negotiation strategies, we also need negotiation protocols that can accommodate negotiators that use different strategies.

A final issue is how to collect credentials that are issued and stored in a distributed manner when authorization decisions need to be made. Again because we are interested in supporting coalitions of organizations that have no shared infrastructure and only limited mutual trust, we believe it is unrealistic to require a centralized credential repository.

1.1. Project Organization

This project has comprised in a single vehicle three independent lines of research by teams at NAI Labs, Brigham Young University (BYU), and the University of Illinois at Urbana-Champaign (UIUC). Each team has advanced the emerging technology of TN. At NAI, we are delivering contributions in the following areas:

1. Distributed credential discovery (algorithms and a credential type system that allows some credentials to be stored with their issuer and some with their subject, while

ensuring credentials can be found to answer authorization questions—this work was done in collaboration with a group at Stanford University operating under a separate DARPA-funded project, SPAWAR N66001-00-C-8015);

2. Policy language design (requirements, as well as identifying a candidate language); and
3. Negotiation strategy (analysis of high-bandwidth covert channels enabling unauthorized access to credential content).

BYU is delivering an analysis and proposed approach to enable resource requestors (clients) to establish trust they need before making a sensitive resource request. This is complimentary to the scenario we discussed above, where the goal of a trust negotiation was to establish trust on the part of the access mediator in the requestor sufficient to satisfy the resource's access-control policy. BYU is also delivering a preliminary draft of a paper about trust negotiation protocols that enable the two parties to negotiate according to different strategies. That work was done in collaboration with UIUC.

Two later versions of the trust negotiation protocols paper are being delivered by UIUC. (Although the BYU group also collaborated on these later drafts, its participation was not funded through the current project.)

The work at NAI Labs has been and continues to be informed and influenced by the work of the subcontractors. However, the subcontractors have conducted their work independently, in a manner reflecting their own views and priorities. The research performed under this project represents partial results and is continuing under separate DARPA-funded follow-on efforts lead by NAI Labs and by BYU.

1.2. Report Organization

Like the project, this report has many parts. The final reports of the subcontractors, as well as several technical reports and publications, are included as attachments. This cover report presents an overview of the work done at NAI Labs. It outlines delivered contributions, and discusses lessons learned and the relationship of work done under the current project to continuing research under the follow-on project. The main body the NAI Labs report begins on page 5.

Attachments 1 and 2 present in greater technical detail the findings of the NAI Labs portion of the project. Attachments 3 and 4 are the final reports of the subcontractors at BYU and UIUC, respectively, which introduce their own project contributions and findings. They introduce Attachments 5 through 8, which are papers written by the subcontractors under the current project.

1.2.1. List of Appendices

Appendix A. “Distributed Credential Chain Discovery in Trust Management,” Ninghui Li, William H. Winsborough, and John C. Mitchell. To appear in: *Eighth ACM Computer and Communications Security Conference*, November, 2001.

Appendix B. “Making Trust Negotiation Realistic: A Suitable Policy Language and the Management of Information about Credential Possession,” William H. Winsborough and Deborah Shands. NAI Labs Technical Report 01-113.

Appendix C. “Advances in Trust Negotiation.” Final report of the Brigham Young subcontract. Kent E. Seamons, PI.

Appendix D. “Advances in Trust Negotiation.” Final report of the University of Illinois subcontract. Marianne Winslett, PI.

Appendix E. “TrustBuilder: Middleware to Enable Trust Negotiation Strategy Interoperability,” Kent Seamons, Marianne Winslett, and Ting Yu. Unpublished report.

Appendix F. “Interoperable Strategies in Automated Trust Negotiation,” T. Yu, M. Winslett, and K. E. Seamons. Accepted submission to the *Eighth ACM Computer and Communications Security Conference* to be held in Philadelphia in November, 2001. (20 pages)

Appendix G. Extended version of : “Interoperable Strategies in Automated Trust Negotiation,” T. Yu, M. Winslett, and K. E. Seamons. Unpublished report. (28 pages)

2. RESEARCH PERFORMED AT NAI LABS

The long-term goal of this research is to develop attribute-based access control mechanisms that provide acceptable scalability and flexibility to dynamic coalitions operating in very large, open networks. NAI Labs is delivering contributions to the solution of two fundamental problems in this area. These are the problems of collecting distributed credentials that prove authorization and of protecting sensitive credential content. By dividing these two difficult problems, we have been able to make significant progress on each of them. In future work, we plan to apply our improved understanding and technique suite to solve the two problems simultaneously. We introduce each of the fundamental problems briefly in this introduction, and then present our contributions to each area in greater detail below, each in its own section.

Today trust negotiation strategies presume that every credential used during a negotiation is held by one participant or the other prior to commencing the negotiation. Yet it is not until negotiation is underway that participants really know which credentials are needed. By its nature, ABAC requires the use of credentials issued to third parties. This is inherent because one of the keys to ABAC's flexibility is that it identifies suitable credential issuers by their attributes. Issuer attributes are documented in those third-party credentials, which we call *supporting credentials*.

Currently, if a negotiation participant does not have a needed supporting credential, it has to break off the negotiation and obtain it out of band. No mechanism for supporting that credential collection has yet been devised. The first fundamental problem NAI Labs is working on is how to integrate and guide the collection of distributed credentials during trust negotiation. We have begun to address the problem by solving a simpler one. Recall from the project overview above that positive authorization decisions are made by finding appropriate credential chains. Ignoring for the time being issues created by credential sensitivity, we have developed automated techniques for collecting credentials that are issued and stored in a distributed manner. These collection techniques are guided by the simultaneous assembly those credential chains. We call this problem distributed credential chain discovery. As we discuss further below, in this area we have completed significant advances that will soon be published in a prominent technical conference.

While our approach to the first problem was to begin by neglecting credential sensitivity, protecting sensitive credentials is the focus of the second fundamental problem that NAI Labs is currently solving. Protecting the contents of credentials from unauthorized disclosure is the basic motivation for trust negotiation. However, most negotiation strategies proposed to date are ineffective in this regard. A concomitant issue is that most prior strategies do not use a realistic policy language. Without an appropriate policy language, the flexibility and scalability of ABAC cannot be achieved. Unfortunately, using a realistic language makes strategy design significantly harder. Nonetheless, it seems fruitless at this stage to design any more negotiation strategies that knowingly fail

to protect credential content or that use unrealistic or toy policy languages. We believe that these problems must be solved simultaneously to have any impact.

Our deliverable regarding this problem is Attachment 2. It presents a snapshot of our work on the two issues of credential protection and language design. It analyzes policy language requirements and examines how negotiators can tell each other their policies for disclosing credentials requested of them without admitting whether they do or do not have the requested credentials. The latter examination is incomplete; our efforts in this area are ongoing under a follow-on DARPA-funded project (SPAWAR N66001-01-C-8005). New difficulties in effectively protecting credential content have continued to emerge since Attachment 2 was written. Some of these late-breaking developments are outlined in our Information Flow discussion below.

The remainder of this report frames contributions to these fundamental problems that NAI Labs is delivering in Attachments 1 and 2.

2.1. Distributed Credential Chain Discovery

Previous trust negotiation research makes the simplifying assumption that, when negotiation begins, the two participants already hold all the credentials that will be used. If one of the participants needs to obtain additional credentials, he must do so outside the context of the negotiation and no support is provided.

For many credentials issued to one of the negotiators, it is reasonable to assume that the negotiator holds a copy. However, as discussed above, supporting credentials, issued to others, will be needed frequently. (Again, this is because part of the flexibility of attribute-based access control derives from the ability to delegate issuing authority to strangers based on the issuer's attributes.) Like credentials issued to the negotiator herself, supporting credentials are presented during trust negotiation so that a complete credential chain can be verified.

Required supporting credentials are identified by the opponent's policy. (The "opponent" is the other negotiator. The relationship resembles that of the forefinger and the "opposable" thumb, and is not normally adversarial.) For negotiators that have no knowledge of one another's policies, it could be very difficult to anticipate before the negotiation which supporting credentials they will need to present to one another. Negotiators know which supporting credentials are relevant only after they begin receiving requests for credentials from the opponent. In principle, if they don't have a copy of a necessary credential, they can deal with this by suspending negotiation, and then resuming it after they have collected the necessary credentials. However, they receive no support for integrating the construction of the credential chain that satisfies a policy with the collection of the credentials that compose the chain.

Setting aside for the time being the problem that the issuers and subjects of supporting credentials may consider those credentials sensitive, we have concentrated in this work on the fact that distributed discovery requires an evaluation procedure that can drive

credential collection. Such a procedure must be goal-oriented in the sense of expending effort only on chains that involve the requester and the access mediator, or its trusted authorities. In very large networks, with distributed storage of thousands or millions of credentials, most of them unrelated to one another, goal-oriented techniques will be crucial. The procedure must also be able to suspend evaluation, issue a request for credentials that could extend partial chains, and then resume evaluation when additional credentials are obtained.

Attachment 1 delivers formally verified and analyzed goal-oriented evaluation algorithms based on a graphical representation of credentials. This graphical representation is ideal for driving credential collection because it makes it easy to suspend and resume evaluation, and to schedule work flexibly. The graph-based evaluation model, presented in Section 3.2, provides a natural structure for organizing the construction of credential chains, while interleaving distributed collection, guided by the chain construction. Very loosely, chains correspond to paths and entities correspond to nodes in this graph. Algorithms, presented in Sections 3.3, 3.4, and 3.5, find chains with worst-case efficiency as good as any known algorithm. Moreover, in the expected case where there are a lot of unrelated credentials, the algorithms avoid wasting time considering them. Thus, even in the centralized case, our goal-oriented algorithms represent a technical contribution.

In addition to the evaluation model and algorithms, Attachment 1 delivers a system for organizing the distributed storage of credentials in a manner that ensures they can be located by trust negotiators. We assume that in general, to find a credential, you need to know the issuer or the subject. We say an entity (subject or issuer) stores the credential if, knowing the entity, you can get the credential, even if it is actually stored by a third party.

Among prior techniques for distributed chain discovery, some store credentials with their issuers and others store credentials with their subjects. Making either assumption, we can find the credentials of a chain by collecting credentials from each successive entity in the chain. In the case where credentials are stored with issuers, we start by collecting credentials from the known, trusted authorities identified by the policy, and work toward the requestor. In the case where credentials are stored with their subjects, we start by collecting from the requestor, and work down the chain in the other direction. The problem with these two approaches is that they are each too restrictive: some credentials should be stored with issuers and some, with subjects. For instances, some credentials may need to be updated frequently, or may be of no interest to the subject. If a business issues a credential to a university that entitles students of that university to special service, the university may not be interested in assisting the arrangement. Conversely, many issuers of large numbers of credentials would become bottlenecks if required to store and make available the credentials they issue. In our example, if a student seeking a special service can provide her own student ID, the university need not actively participate.

Attachment 1 presents a novel way to ensure it is possible to find all credentials relevant to a given authorization decision while allowing some credentials to be stored with their subjects and some with their issuers. It introduces a type system for credentials that introduces significant flexibility in the location of credential storage. Roughly speaking, if credentials are well typed, you can find chains by starting at both ends and working toward the middle. Section 4 introduces the type system and proves that it ensures relevant credentials can be found when needed to support authorization decisions. Section 4.4 presents suggestions for making typing, as well as natural language descriptions of attribute meanings, accessible and uniform by using a technique similar to XML namespaces.

Finally, Section 2 of Attachment 1 presents a new credential and policy language that may in future work lead us to revise our language choice, which we discuss in attachment 2. We return to this issue in the next section.

2.1.1. Lessons Learned and Future Work in Distributed Credential Discovery

Section 5.2 shows that credential chain discovery for our language is log-space P-complete. Unfortunately, this means that in the worst case, the problem is probably not amenable to speedup through parallelization. In effect, this means that showing that some policies are satisfied will probably always be an expensive proposition. The expectation is that, while it may be difficult to exclude such policies from a sufficiently expressive language, the policies that occur naturally will be unlikely to require excessive evaluation effort.

As mentioned above, the work presented in Attachment 1 ignores the problem of credential sensitivity, which is the motivation for performing trust negotiation as part of attribute-based access control. Integrating management of credential sensitivity and distributed credential discovery remains a matter for future work.

2.2. Trust Negotiation: Analyzing Language Requirements and Information Flow

Attachment 2 presents partial results of our on-going efforts to design a realistic trust negotiation strategy. It focuses on two issues. The first is the design or adaptation of a suitable policy and credential language. Section 4.1 of the attachment critiques policy languages used in prior trust negotiation strategies and shows why each fails to meet key policy language requirements, which are also presented. Section 4.2 discusses the suitability of Delegation Logic, which we tentatively selected for use in future strategy design. As seen in Section 2 of Attachment 1, we have recently become involved in on-going work lead by researchers at Stanford University that may yield an even better language for our purpose. That work seeks, among other things, to simplify the somewhat difficult notation of DL by incorporating SDSI-like notations. It also seeks to develop evaluation models that better support the interleaving of credential collection

with proof construction. The language used in attachment 1 does not, however, allow attributes to have fields (*i.e.*, parameters), such as age or credit limit. For this reason, at present, the most suitable existing language for trust negotiation that we know of remains Delegation Logic.

The second issue concerns the management of credential-content information flow. The goal of this line of work is to create strategies in which negotiators respond to requests for credentials in a way that does not disclose which credentials they have before the requestor has demonstrated that they are authorized to receive that information. By this criterion, all previous negotiation strategies have failed, with the exception of the eager strategy, which has other problems (too many irrelevant credentials flow). Correcting this problem is a central, on-going goal of this research.

Our basic technique for controlling information about the contents of credentials a negotiator possesses is to enforce policies that govern acknowledging possession or non-possession of sensitive credentials. (Sometimes having a credential is sensitive; sometimes not having one is sensitive.) We call these policies *acknowledgement policies*, in contrast to access-control policies, which also must be satisfied before a credential is transmitted. The idea is that once the appropriate acknowledgement policies have been satisfied, the access control policies of individual credentials can flow, which implicitly acknowledges possession of the requested credentials. Acknowledgement policies are transmitted even when the requested credentials are not held, so their transmission does not indicate possession of any credentials.

We assume that every credential has a credential type, such as “drivers license” or “letter of credit,” which identifies the variety of credential it is. Credentials may also have data fields containing values, which could be sensitive, such as age or credit limit.

In preliminary design work described in Section 3.3 of Attachment 2, we noted the importance of a negotiator always responding to a request for credentials of type p in the same way, no matter how many (zero or more) credentials he has of that type. In particular, we observed that the number and content of policies transmitted in the initial response to a request (the acknowledgement policies) should not depend on the credentials the transmitter actually holds. Thus, we proposed associating an acknowledgement policy with each credential type. When a request for credentials of a given type is received, the acknowledgement policy is transmitted. Once that policy is satisfied, the negotiator either discloses that he has no credentials of that type, or transmits access control policies for each credential he has of the type.

Section 5 of Attachment 2 presents an abstract analysis aimed at either justifying or finding fault with this design. As far as it goes, it seems to justify several of our design choices. However, additional issues not discussed there have surfaced concerning the management of credential-content information flow. We have identified two additional refinements that should be made before completing a formal design specification. First, we want to make use of constraints on credential contents in credential requests to reduce

the number of unnecessary credentials and policies that flow, and thereby to make trust negotiation more efficient. Second, and more critically, we need to ensure that acknowledgement of credential possession occurs at a sufficiently fine granularity to avoid forcing inappropriate compromises between confidentiality and availability. Both of these refinements lead to a need for acknowledgement policies that depend on credential field values, as well as on the credential type.

Although we have begun to work out a design along these lines, we came to appreciate the importance of these refinements too late to be able to develop a design specification based on them during the current project. However, we believe that coming to appreciate their importance has been a significant advance, which will lead to the development of a family of high-quality designs as part of the follow-on project. The remainder of this section explains why these refinements now appear to be so important.

When a transmitted policy is received by a negotiator, it is taken as a request for credentials that satisfy the policy. If credential requests constrain the field values in the requested credentials, then when a negotiator's behavior discloses, implicitly or explicitly, that a request cannot be satisfied, contents of available credentials can be inferred based on the values specified in the request. Acknowledgement policies based on credential type can be used to ensure that these disclosures are only made to appropriate opponents, since any reasonable negotiation strategy will acknowledge request unsatisfiability only after the acknowledgement policy is satisfied. However, relying on the same acknowledgement policy to control acknowledgement of all credentials of a given type, no matter what values they contain, leads to a coarse-grained system of control. It means that credentials containing very sensitive values and credentials containing values that are not sensitive at all must be governed by the same acknowledgement policy, just because they have the same credential type. The result is either that the content of the sensitive credential is inadequately protected, or that the non-sensitive credential is less available than it should be, making negotiation success less common than it should reasonably be.

One quick fix to this problem would be to require that credential types be different whenever sensitivity levels are different. However, it would artificially force the credential type to contain information that is more naturally carried in one of the credential's fields. For instance, consider a simple membership credential. There will already be a field containing the credential's issuer, and the namespace would be needlessly cluttered if required to encode the issuer in the type. Yet the degree of sensitivity associated with a membership certainly depends on the issuing organization. In another example, a letter of credit of ten million dollars is likely to have a very different degree of sensitivity than one of ten thousand. Yet to require a distinction among the types of these credentials would be arbitrary and unenforceable.

It should also be noted that this granularity problem is not solely a consequence of the design decision that credential requests should be allowed to constrain credential contents. We now argue that when access control policies flow, they may disclose

credential contents as well. The implication is that, independent of constraints in requests, relying on the same acknowledgement policy to control acknowledgement of all credentials of a given type, no matter what values they contain, once again leads to an unacceptably coarse-grained system of control.

To see that access control policies can disclose contents, consider the following. It is reasonable to assume that credential access control policies used for the same credential by different negotiators will often be similar, perhaps even standardized. In particular, even among credentials of the same type, these standardized policies may often differ according to the values of parameters. For instance, a letter of credit of ten million dollars is likely to have a very different access control policy than one of ten thousand. Thus, when the access control policy flows, its recipient can infer something about the size of the credit line. Once again, if the same acknowledgement policy is used for all letter-of-credit credentials, then either the information that the negotiator has a ten million dollar line of credit is inadequately protected, or the smaller letter is governed unnecessarily strictly, occasionally even needlessly preventing successful negotiation.

We are currently designing a negotiation strategy in which acknowledgement of different field values is governed differentially. Part of the problem is that a negotiator can't send a separate acknowledgement policy for each of his credentials without disclosing how many credentials he has, or without hinting at the values of those credentials. We get around this hazard by allowing each negotiator to establish a separate acknowledgement policy for each field value within each credential type, as well as for the credential type itself. The design requires a default value policy to be associated with each field, and then allows special values to be given special policies. Note that these values should have no correlation with the values of credentials the negotiator actually holds.

(While a negotiator clearly cannot generate policies for all field values for all credential types, it is in the negotiator's interest to provide policies for values and types whose possession or non-possession would be sensitive, even when he doesn't have (or, respectively, does have) that credential. This is because treating information as sensitive when you actually have nothing to hide reduces the information content of treating something as sensitive. We anticipate credential issuers publishing suggested acknowledgement policies that negotiators can use, whether they have the credentials or not, thereby making it easy for individuals to respond uniformly whether or not they have the credential.)

When a request for credentials arrives, the relevant field value acknowledgement policies all flow. As acknowledgement policies for field values occurring in particular credentials are satisfied, the access control policies for those credentials can flow. In this way, the policies transmitted when a request arrives don't reveal which credentials are actually held, yet the acknowledgement policies that are enforced are appropriate to each individual credential, yielding the desired fine-grained control.

2.2.1. Lessons Learned and Future Work in Information Flow

Our original plan was to develop algorithms supporting the parsimonious negotiation strategy. (See Section 2.2.2 of Attachment 2 for a summary of this strategy.) As we began to study information flow in this strategy, however, we realized that the parsimonious strategy would probably never be acceptable to would-be adopters. The problems with information flow in this strategy are presented in Section 3.2 of Attachment 2. In light of this realization, we dropped further work on the parsimonious strategy from our plan and focused instead on design requirements and principles for better managing information flow in trust negotiation.

Our experience with this design problem has illustrated the general difficulty of managing covert channels. It is generally acknowledged that efforts to eliminate all covert channels in access control systems are unlikely to succeed. Yet covert channels that are sufficiently high-bandwidth demand attention. In our work, for instance, many prior negotiation strategies were entirely unsuccessful in, or even made no effort aimed at, preventing the opponent determining exactly which credentials a negotiator holds. By comparison, the issues discussed above are much more subtle.

We have avoided expending effort on issues that seem unlikely to be central to trust negotiation's viability as a technology. For instance, we have no plans at this time to support special field value acknowledgement policies for combinations of field values. The acknowledgement policy for a combination of field values will just be the conjunction of acknowledgement policies for the individual field values. If stronger combination policies turn out to be important in practice, they can be added later.

Potential covert channels have sometimes not been immediately evident. Unfortunately, it took us several months to notice that the content of the access control policy itself could reveal important sensitive features of the credential it governs.

The reader of Attachment 2 Section 5 will see that organizing the principles that underlie a successful design is not a simple matter. This probably is why prior strategies, as well as several of our own early attempts, were unsuccessful.

Although we have performed substantial analysis in this area and developed several design principles that seem solid, we will not complete this design under the current contract, as insufficient time and funding remain. However, this work continues, uninterrupted, under the DARPA-funded follow-on contract (SPAWAR N66001-01-C-8005), and we expect to have a completed design this Fall.

2.3. Conclusions

In this preliminary study, we have taken several important steps toward meeting our long-term goal of developing access control systems that are highly scalable, yet fine-grained, making them suitable for the demands of dynamic coalitions. In the area of distributed credential discovery, we have provided algorithms and a credential type system that allow some credentials to be stored with their issuer and some with their

subject, while ensuring that credentials can be located as needed to answer authorization questions. In the area of policy language design, we have identified basic requirements for ABAC policy languages. We found that none of the policy languages used in existing trust negotiation strategies meet these requirements, and we identified Delegation Logic (DL) as a candidate language that meets our basic requirements. In our work toward designing a realistic negotiation strategy, we have analyzed existing strategies from the point of view of whether they successfully control access to credential content and information about which credentials a negotiator holds. The important ones all have high-bandwidth covert channels that enable unauthorized access to credential content. We have developed design principles that seem to close the covert channels we have identified and we are in the process of organizing a design specification. Our work in these and other fundamental ABAC technologies continues uninterrupted in a DARPA-funded follow-on project.

Distributed Credential Chain Discovery in Trust Management (Extended Abstract) *

Ninghui Li
Department of Computer
Science, Gates 4B
Stanford University
Stanford, CA 94305-9045
ninghui.li@cs.stanford.edu

William H. Winsborough
NAI Labs
Network Associates, Inc.
3060 Washington Road
Glenwood, MD 21738
William.Winsborough@NAI.com

John C. Mitchell
Department of Computer
Science, Gates 4B
Stanford University
Stanford, CA 94305-9045
mitchell@cs.stanford.edu

ABSTRACT

We give goal-oriented algorithms for discovering credential chains in RT_0 , a role-based trust-management language introduced in this paper. The algorithms search credential graphs, a representation of RT_0 credentials. We prove that evaluation based on reachability in credential graphs is sound and complete with respect to the set-theoretic semantics of RT_0 . RT_0 is more expressive than SDSI 2.0, so our algorithms can perform chain discovery in SDSI 2.0, for which existing algorithms in the literature either are not goal-oriented or require using specialized logic-programming inferencing engines. Being goal-oriented enables our algorithms to be used when credential storage is distributed. We introduce a type system for credential storage that guarantees well-typed, distributed credential chains can be discovered.

1. INTRODUCTION

Several trust-management systems have been proposed in recent years, *e.g.*, SPKI/SDSI [10], PolicyMaker [3, 4], KeyNote [2], Delegation Logic [15]. These systems are based on the notion of delegation, whereby one entity gives some of its authority to other entities. The process of making access control decisions involves finding a delegation chain from the source of authority to the requester. Thus, a central problem in trust management is to determine whether such a chain exists and, if so, to find it. We call this the *credential chain discovery problem*, by contrast with the *certificate chain discovery problem*, which is concerned with X.509 certificates [9]. Credentials in trust management generally have more complex meanings than simply binding names to public keys, and a credential chain is often a graph, rather than a linear path. The goal of this paper is to address the

credential chain discovery problem (the *discovery problem* for short) in such systems.

Almost all existing work addressing the discovery problem assumes that potentially relevant credentials are all gathered in one place. This is at odds with the tenet of trust management—decentralized control; systems that use trust management typically issue and often store credentials in a distributed manner. This raises some nontrivial questions.

EXAMPLE 1. *A fictitious Web publishing service, EPub, offers a discount to preferred customers of its parent organization, EOrg. EOrg issues a credential to the ACM stating that ACM members are preferred customers. Combining it with Alice's ACM membership credential yields a two-credential chain that proves Alice is a preferred customer. This is a linear chain; the subject of the credential issued by EOrg, ACM, is the issuer of the credential issued to Alice.*

These two credentials must be collected to construct the chain. The question we take up is where they should be stored to enable that collection. We say an entity *A* stores a credential if we can find the credential once we know *A*. Some other entity, such as a directory, may actually house the credential on *A*'s behalf. Also, by storing a credential, we mean storing and providing access to the credential.

Given this definition of storing, to be useful, a credential must be stored with its issuer or with its subject. If both credentials in example 1 are stored with their subject, we can find them by obtaining the first credential from Alice, and using the issuer of that credential, ACM, to obtain the second. A disadvantage of this strategy is that it requires the ACM to store all the credentials authorizing ACM members. This makes the ACM a bottleneck. Also, some issuers may not entrust credentials to their subjects. If instead both credentials are stored with their issuers, the ACM has to store and provide all membership ids, again making it a bottle neck, and potentially causing broad search fan-out.

In the example, the ideal arrangement is to store one credential with EOrg and the other with Alice. The chain can then be discovered by working from these two ends towards the chain's middle. No prior credential discovery system supports this, probably because subject- and issuer-storage cannot be intermixed arbitrarily: in our example, if both credentials are stored exclusively by the ACM, the chain cannot be found. This is because in many decentralized systems, it is impossible or prohibitively expensive for one entity to enumerate all other entities in the systems. For all practical purposes, in such a system, if one can't find a

*A full version of this paper is available at: <http://crypto.stanford.edu/~ninghui/papers/disc.pdf>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CCS'01, November 5-8, 2001, Philadelphia, Pennsylvania, USA.
Copyright 2001 ACM 1-58113-385-5/01/0011 ...\$5.00.

credential chain without contacting every entity, one can't find it at all. In this paper, we introduce a credential typing system that constrains storage enough to ensure chains can be found by starting at their two ends and working inward.

The credential chain introduced in example 1 illustrates only the simplest case that we address. Some trust management systems, such as SDSI and Delegation Logic, allow what we call *attribute-based delegation*, that is the delegation of attribute authority to entities having certain attributes.

EXAMPLE 2. *EPub offers another discount to university students, and delegates the authority over the identification of students to entities that are accredited universities.*

Attribute-based delegation is achieved in SDSI through linked names, and in Delegation Logic through dynamic threshold structures and through conditional delegations. Systems that support attribute-based delegation promise high flexibility and scalability. However they significantly complicate the structure and discovery of credential chains.

Beyond storing credentials where they can be found, distributed discovery also requires an evaluation procedure that can drive credential collection. Such a procedure must be goal-oriented in the sense of expending effort only on chains that involve the requester and the access mediator, or its trusted authorities. In the Internet, with distributed storage of millions of credentials, most of them unrelated to one another, goal-oriented techniques will be crucial. The procedure must also be able to suspend evaluation, issue a request for credentials that could extend partial chains, and then resume evaluation when additional credentials are obtained. Existing evaluation procedures for SDSI and for Delegation Logic are either not goal-oriented, or do not support this alternation between collection and evaluation steps.

As a concrete foundation for discussing the discovery problem, we introduce a trust-management language, RT_0 , which supports attribute-based delegation and subsumes SDSI 2.0 (the "SDSI" part of SPKI/SDSI 2.0 [10]). We provide goal-oriented evaluation algorithms based on a graphical representation of RT_0 credentials. This representation is ideal for driving credential collection because it makes it easy to suspend and resume, and to schedule work flexibly. Even in the centralized case, goal-orientation is an advantage when the credential pool is very large and contains many credentials that are unrelated. We also show how to use our algorithms to perform goal-oriented chain discovery for SDSI 2.0.

The rest of this paper is organized as follows. In section 2, we present the syntax and a set-theoretic semantics for RT_0 . In section 3, we present goal-oriented, graph-based algorithms for centralized chain discovery in RT_0 , and show how to apply them to SDSI as well. We prove that the graph-based notion of credential chains is sound and complete with respect to the semantics for RT_0 . In section 4, we study chain discovery in the distributed case. We present a notion of well-typed credentials and prove that chains of well-typed credentials can always be discovered. In section 5, we discuss future directions and some related work. We conclude in section 6.

2. A ROLE-BASED TRUST-MANAGEMENT LANGUAGE

This section introduces RT_0 , the first (and the simplest) in a series of role-based trust-management languages we are developing. We present RT_0 's syntax, discuss its intended

meaning, and compare it to SDSI. Then we give a formal semantics.

2.1 The Language RT_0

The constructs of RT_0 include entities, role names, and roles. Typically, an *entity* is a public key, but could also be, say, a user account. Entities can issue credentials and make requests. RT_0 requires that each entity can be uniquely identified and that one can determine which entity issued a particular credential or a request. In this paper, we use A , B , and D to denote entities. A *role name* is an identifier, say, a string. We use r , r_1 , r_2 , etc., to denote role names. A *role* takes the form of an entity followed by a role name, separated by a dot, e.g., $A.r$ and $B.r_1$. The notion of roles is central in RT_0 . A role has a value that is a set of entities who are members of this role. Each entity A has the authority to define who are the members of each role of the form $A.r$. A role can also be viewed as an attribute. An entity is a member of a role if and only if it has the attribute identified by the role. In RT_0 , an access control permission is represented as a role as well. For example, the permission to shut down a computer can be represented by a role $OS.shutdown$.

There are four kinds of credentials in RT_0 , each corresponding to a different way of defining role membership:

- *Type-1:* $A.r \leftarrow B$

A and B are (possibly the same) entities, and r is a role name.

This means that A defines B to be a member of A 's r role. In the attribute-based view, this credential can be read as B has the attribute $A.r$, or equivalently, A says that B has the attribute r .

- *Type-2:* $A.r \leftarrow B.r_1$

A and B are (possibly the same) entities, and r and r_1 are (possibly the same) role names.

This means that A defines its r role to include all members of B 's r_1 role. In other words, A defines the role $B.r_1$ to be more powerful than $A.r$, in the sense that a member of $B.r_1$ can do anything that the role $A.r$ is authorized to do. Such credentials can be used to define role-hierarchy in Role-Based Access Control (RBAC) [16]. The attribute-based reading of this credential is: if B says that an entity has the attribute r_1 , then A says that it has the attribute r . In particular, if r and r_1 are the same, this is a delegation from A to B of authority over r .

- *Type-3:* $A.r \leftarrow A.r_1.r_2$

A is an entity, and r , r_1 , and r_2 are role names. We call $A.r_1.r_2$ a *linked role*.

This means that $members(A.r) \supseteq members(A.r_1.r_2) = \bigcup_{B \in members(A.r_1)} members(B.r_2)$, where $members(e)$ represents the set of entities that are members of e . The attribute-based reading of this credential is: if A says that an entity B has the attribute r_1 , and B says that an entity D has the attribute r_2 , then A says that D has the attribute r . This is attribute-based delegation: A identifies B as an authority on r_2 not by using (or knowing) B 's identity, but by another attribute of B (viz., r_1). If r and r_2 are the same, A is delegating

its authority over r to anyone that A believes to have the attribute r_1 .

- *Type-4:* $A.r \leftarrow f_1 \cap f_2 \cap \dots \cap f_k$

A is an entity, k is an integer greater than 1, and each f_j , $1 \leq j \leq k$, is an entity, a role, or a linked role starting with A . We call $f_1 \cap f_2 \cap \dots \cap f_k$ an *intersection*.

This means that $\text{members}(A.r) \supseteq (\text{members}(f_1) \cap \dots \cap \text{members}(f_k))$. The attribute-based reading of this credential is: anyone who has all the attributes $f_1, \dots, f_k$ also has the attribute $A.r$.

A *role expression* is an entity, a role, a linked role, or an intersection. We use e, e_1, e_2 , etc, to denote role expressions. By contrast, we use $f_1, \dots, f_k$ to denote the intersection-free expressions occurring in intersections. All credentials in RT_0 take the form, $A.r \leftarrow e$, where e is a role expression. Such a credential means that $\text{members}(A.r) \supseteq \text{members}(e)$, as we formalize in section 2.2 below. We say that this credential *defines* the role $A.r$. (This choice of terminology is motivated by analogy to name definitions in SDSI, as well as to predicate definitions in logic programming.) We call A the *issuer*, e the *right-hand side*, and each entity in $\text{base}(e)$ a *subject* of this credential, where $\text{base}(e)$ is defined as follows: $\text{base}(A) = \{A\}$, $\text{base}(A.r) = \{A\}$, $\text{base}(A.r_1.r_2) = \{A\}$, $\text{base}(f_1 \cap \dots \cap f_k) = \text{base}(f_1) \cup \dots \cup \text{base}(f_k)$.

EXAMPLE 3. Combining examples 1 and 2, EPub offers a special discount to anyone who is both a preferred customer of EOrg and a student. To identify legitimate universities, EPub accepts accrediting credentials issued by the fictitious Accrediting Board for Universities (ABU). The following credentials prove Alice is eligible for the special discount:

$$\left\{ \begin{array}{l} \text{EPub.spdiscount} \leftarrow \text{EOrg.preferred} \cap \text{EPub.student}, \\ \text{EOrg.preferred} \leftarrow \text{ACM.member}, \\ \text{ACM.member} \leftarrow \text{Alice}, \\ \text{EPub.student} \leftarrow \text{EPub.university.stuID}, \\ \text{EPub.university} \leftarrow \text{ABU.accredited}, \\ \text{ABU.accredited} \leftarrow \text{StateU}, \\ \text{StateU.stuID} \leftarrow \text{Alice} \end{array} \right\}$$

Readers familiar with Simple Distributed Security Infrastructure (SDSI) [8, 10] may notice the similarity between RT_0 and SDSI's name certificates. Indeed, our design is heavily influenced by existing trust-management systems, especially SDSI and Delegation Logic (DL) [15]. RT_0 can be viewed as an extension to SDSI 2.0 or a syntactically sugared version of a subset of DL. The arrows in RT_0 credentials are the reverse direction of those in SPKI/SDSI. We choose to use this direction to be consistent with an underlying logic programming reading of credentials and with directed edges in credential graphs, introduced below in section 3. In addition, RT_0 differs from SDSI 2.0 in the following two aspects.

First, SDSI allows arbitrarily long linked names, while we allow only length-2 linked roles. There are a couple of reasons for this design. We are not losing any expressive power; one can always break up a long chain by introducing additional roles and credentials. Moreover, it often makes sense to break long chains up, as doing so creates more modular policies. If A wants to use $B.r_1.r_2 \dots r_k$ in its credential, then $B.r_1.r_2 \dots r_{k-1}$ must mean something to A ; otherwise, why would A delegate power to members of $B.r_1.r_2 \dots r_{k-1}$? Having to create a new role makes A

think about what $B.r_1.r_2 \dots r_{k-1}$ means. Finally, restricting lengths of linked roles simplifies the design of algorithms for chain discovery.

Second, SDSI doesn't have RT_0 's type-4 credentials, and so RT_0 is more expressive than the current version of SDSI 2.0. Intersections and threshold structures (e.g., those in [10]) can be used to implement one another. Threshold structures may appear in name certificates according to [10] and earlier versions of [11]. This is disallowed in [8] and the most up-to-date version of [11], because threshold structures are viewed as too complex [8]. Intersections provide similar functionality with simple and clear semantics.

2.2 The Semantics of RT_0

This section presents a non-operational semantics of RT_0 . Given a set $\mathcal{C}$ of RT_0 credentials, we define a map $\mathcal{S}_{\mathcal{C}} : \text{Roles} \rightarrow \wp(\text{Entities})$, where $\wp(\text{Entities})$ is the power set of Entities. $\mathcal{S}_{\mathcal{C}}$ is given by the least solution to a system of set inequalities that is parameterized by a finite, set-valued function, $\text{rmem} : \text{Roles} \rightarrow \wp(\text{Entities})$. That is, the semantics is the least such function that satisfies the system, where the ordering is pointwise subset. We use a least fixpoint so as to resolve circular role dependencies. To help construct the system of inequalities, we extend rmem to arbitrary role expressions (whose domain we denote by RoleExpressions) through the use of an auxiliary semantic function, $\text{expr}_{\text{rmem}} : \text{RoleExpressions} \rightarrow \wp(\text{Entities})$ defined as follows:

$$\begin{aligned} \text{expr}_{\text{rmem}}(B) &= \{B\} \\ \text{expr}_{\text{rmem}}(A.r) &= \text{rmem}(A.r) \\ \text{expr}_{\text{rmem}}(A.r_1.r_2) &= \bigcup_{B \in \text{rmem}(A.r_1)} \text{rmem}(B.r_2) \\ \text{expr}_{\text{rmem}}(f_1 \cap \dots \cap f_k) &= \bigcap_{1 \leq j \leq k} \text{expr}_{\text{rmem}}(f_j) \end{aligned}$$

We now define $\mathcal{S}_{\mathcal{C}}$ to be the least value of rmem satisfying the following system of inequalities:

$$\{ \text{expr}_{\text{rmem}}(e) \subseteq \text{rmem}(A.r) \mid A.r \leftarrow e \in \mathcal{C} \}.$$

As with rmem , we use expr to extend $\mathcal{S}_{\mathcal{C}}$ to role expressions, writing $\text{expr}_{\mathcal{S}_{\mathcal{C}}}(e)$ for the members of role expression e .

The least solution to such a system can be constructed as the limit of a sequence $\{\text{rmem}_i\}_{i \in \mathcal{N}}$, where $\mathcal{N}$ is the set of natural numbers, and where for each i , $\text{rmem}_i : \text{Roles} \rightarrow \wp(\text{Entities})$. The sequence is defined inductively by taking $\text{rmem}_0(A.r) = \emptyset$ for each role $A.r$ and by defining rmem_{i+1} so that for each role $A.r$,

$$\text{rmem}_{i+1}(A.r) = \bigcup_{A.r \leftarrow e \in \mathcal{C}} \text{expr}_{\text{rmem}_i}(e).$$

The function that relates the values of $\{\text{rmem}_i\}_{i \in \mathcal{N}}$ is monotonic, because the operators used to construct it ($\cap$ and $\cup$) are monotonic. Furthermore, $\text{Roles} \rightarrow \wp(\text{Entities})$ is a complete lattice. So this sequence is known to converge to the function's least fixpoint, which is clearly also the least solution to the inequalities. (As the lattice is finite, convergence takes place finitely.) Thus, the least solution exists and is easily constructed. For instance, referring to example 3 and showing only changes in the function's value, successive values of rmem_i have: for $i = 1$, $\text{ABU.accredited} = \{\text{StateU}\}$, $\text{StateU.stuID} = \{\text{Alice}\}$, $\text{ACM.member} = \{\text{Alice}\}$; for $i = 2$, $\text{EPub.university} = \{\text{StateU}\}$, $\text{EOrg.preferred} = \{\text{Alice}\}$; for $i = 3$, $\text{EPub.student} = \{\text{Alice}\}$; for $i = 4$, $\text{EPub.spdiscount} = \{\text{Alice}\}$, where they stabilize.

3. CENTRALIZED CHAIN DISCOVERY

Given a set of credentials $\mathcal{C}$ in RT_0 , three important kinds of queries are:

1. Given a role $A.r$, determine its member set, $\mathcal{S}_C(A.r)$;
2. Given an entity D , determine all the roles it belongs to, *i.e.*, all role $A.r$'s such that $D \in \mathcal{S}_C(A.r)$;
3. Given a role $A.r$ and an entity D , determine whether $D \in \mathcal{S}_C(A.r)$.

In this section, we study credential chain discovery for RT_0 when credentials are centralized. We give goal-oriented algorithms for answering the above three kinds of queries.

3.1 Algorithm Requirements

Chain discovery in RT_0 shares two key problem characteristics with discovery in SDSI: linked names give credential chains a non-linear structure and role definitions can be cyclic. Cyclic dependencies must be managed to avoid non-termination. Clarke *et al.* [8] have given an algorithm for chain discovery in SPKI/SDSI 2.0. Their algorithm views each certificate as a rewriting rule and views discovery as a term-rewriting problem. It manages cyclic dependency by using a bottom-up approach—it performs a closure operation over the set of all credentials before it finds one chain. This may be suitable when large numbers of queries are made about a slowly changing credential pool of modest size. However, as the frequency of changes to the credential pool (particularly deletions, such as credential expirations or revocations) approaches the frequency of queries against the pool, the efficiency of the bottom-up approach deteriorates rapidly, particularly when pool size is large.

Li [14] gave a 4-rule logic program to calculate meanings of SDSI credentials. Cyclic dependencies are managed by using XSB [17] to evaluate the program. XSB's extension table mechanism avoids non-termination problems to which other Prolog engines succumb. Yet, for many trust-management applications, this solution is excessively heavy-weight. Moreover, in its current form, the resulting evaluation mechanism cannot be used to drive credential collection.

As discussed in section 1, because we seek techniques that work well when the credential pool is distributed or changes frequently, we require chain discovery algorithms that are goal-directed and that can drive the collection process. They also must support interleaving credential collection and chain construction (*i.e.*, evaluation) steps.

We meet these requirements by providing graph-based evaluation algorithms. Credentials are represented by edges. Chain discovery is performed by starting at the node representing the requester, or the node representing the role (permission) to be proven, or both, and then traversing paths in the graph trying to build an appropriate chain. In addition to being goal-directed, this approach allows the elaboration of the graph to be scheduled flexibly. Also, the graphical representation of the evaluation state makes it relatively straightforward to manage cyclic dependencies. To our knowledge, our algorithms are the first to use a graphical representation to handle linked roles.

3.2 A Graph Representation of Credentials

We define a directed graph, which we call a *credential graph*, to represent a set of credentials and their meanings. Each node in the graph represents a role expression occurring in a credential in $\mathcal{C}$. Every credential $A.r \leftarrow e \in \mathcal{C}$

contributes an edge $e \rightarrow A.r$.¹ (This holds for credentials of all types.) The destinations of these edges are roles. Edges are also added whose destinations are linked roles and intersections. We call these *derived* edges because their inclusion come from the existence of other, semantically related, paths in the graph.

DEFINITION 1 (CREDENTIAL GRAPH). *For a set of credentials $\mathcal{C}$, the corresponding credential graph is given by $G_C = \langle N_C, E_C \rangle$ where N_C and E_C are defined as follows.*

$$N_C = \bigcup_{A.r \leftarrow e \in \mathcal{C}} \{A.r, e\}.$$

E_C is the least set of edges over N_C that satisfies the following three closure properties:

Closure Property 1: *If $A.r \leftarrow e \in \mathcal{C}$, then $e \rightarrow A.r \in E_C$.*

Closure Property 2: *If $B.r_2, A.r_1.r_2 \in N_C$ and there is a path $B \xrightarrow{*} A.r_1$ in E_C , then $B.r_2 \rightarrow A.r_1.r_2 \in E_C$; we say that this edge is derived from the path $B \xrightarrow{*} A.r_1$.*

Closure Property 3: *If $D, f_1 \cap \dots \cap f_k \in N_C$ and for each $j \in [1..k]$ there is a path $D \xrightarrow{*} f_j$, then $D \rightarrow f_1 \cap \dots \cap f_k \in E_C$; we say that this edge is derived from the paths $D \xrightarrow{*} f_j$, for $j \in [1..k]$.*

This definition can be made effective by inductively constructing a sequence of edge sets $\{E_C^i\}_{i \in \mathcal{N}}$ whose limit is E_C . We take $E_C^0 = \{e \rightarrow A.r \mid A.r \leftarrow e \in \mathcal{C}\}$ and construct E_C^{i+1} from E_C^i by adding one edge according to either closure property 2 or 3. Since $\mathcal{C}$ is finite, we do not have to worry about scheduling these additions. At some finite stage, no more edges will be added, and the sequence converges to E_C .

THEOREM 1 (SOUNDNESS). *Given an entity D and a role expression e , if there is a path $D \xrightarrow{*} e$ in E_C , then $D \in \text{expr}_{S_C}(e)$.*

PROOF. The proof is by induction on the steps of the construction of $\{E_C^i\}_{i \in \mathcal{N}}$ shown above. We prove an induction hypothesis that is slightly stronger than the theorem: For each $i \in \mathcal{N}$ and for any role expressions e_1 and e , if there is a path $e_1 \xrightarrow{*} e$ in E_C^i , then $\text{expr}_{S_C}(e_1) \subseteq \text{expr}_{S_C}(e)$.

We show the base case by using a second, inner induction on the length of the path $e_1 \xrightarrow{*} e$ in E_C^0 . The inner base case, in which $e_1 = e$, is trivial; we consider the step. Suppose $(e_1 \xrightarrow{*} e) = (e_1 \xrightarrow{*} e_2 \rightarrow e)$. Because each edge in E_C^0 corresponds to a credential, we have $e \leftarrow e_2 \in \mathcal{C}$. It follows that $\text{expr}_{S_C}(e_2) \subseteq \text{expr}_{S_C}(e)$, by definition of $\mathcal{S}_C$. The induction assumption gives us $\text{expr}_{S_C}(e_1) \subseteq \text{expr}_{S_C}(e_2)$, so $\text{expr}_{S_C}(e_1) \subseteq \text{expr}_{S_C}(e)$.

We prove the step by again using an inner induction on the length of $e_1 \xrightarrow{*} e$, which we now assume is in E_C^{i+1} . Again the basis is trivial. For the step, we decompose $e_1 \xrightarrow{*} e$ into $e_1 \xrightarrow{*} e_2 \rightarrow e$. There are three cases, depending on which closure property introduced the edge $e_2 \rightarrow e$.

case 1: When $e_2 \rightarrow e$ is introduced by closure property 1, the argument proceeds along the same lines as the base case, using the inner induction hypothesis on $e_1 \xrightarrow{*} e_2$ to derive $\text{expr}_{S_C}(e_1) \subseteq \text{expr}_{S_C}(e_2)$.

¹While long, lefthand arrows ($\leftarrow$) represent credentials, short, righthand arrows ($\rightarrow$) represent edges, and short, righthand arrows with stars ($\xrightarrow{*}$) represent paths, which consist of zero or more edges.

case 2: When $e_2 \rightarrow e$ is introduced by closure property 2, e has the form $A.r_1.r_2$, e_2 has the form $B.r_2$, and there is a path $B \xrightarrow{*} A.r_1$ in E_C^i . The outer induction hypothesis gives us $\text{expr}_{S_C}(B) \subseteq \text{expr}_{S_C}(A.r_1)$, i.e., $B \in S_C(A.r_1)$. The inner induction hypothesis gives us $\text{expr}_{S_C}(e_1) \subseteq \text{expr}_{S_C}(B.r_2)$. Together with the definition of expr for $A.r_1.r_2$, these imply $\text{expr}_{S_C}(e_1) \subseteq \text{expr}_{S_C}(e)$, as required.

case 3: When $e_2 \rightarrow e$ is introduced by closure property 3, e has the form $f_1 \cap \dots \cap f_k$, $e_2 = e_1$ is an entity D (because entity nodes have no incoming edges), and there are paths $D \xrightarrow{*} f_j$ in E_C^i for each $j \in [1..k]$. The outer induction hypothesis gives us $D \in \text{expr}_{S_C}(f_j)$ for $j \in [1..k]$; therefore, $\text{expr}_{S_C}(e_1) \subseteq \text{expr}_{S_C}(e)$. $\square$

THEOREM 2 (COMPLETENESS). *For any role, $A.r$, $D \in S_C(A.r)$ implies there exists a path $D \xrightarrow{*} A.r$ in E_C .*

The proofs for this and other theorems are omitted due to space limitation; they can be found in the full version of this paper.

Together, Theorems 1 and 2 tell us that we can answer each of the queries enumerated at the top of this section by consulting the credential graph. The rest of this section gives algorithms for constructing subgraphs that enable us to answer such questions without constructing the entire graph. As we have seen, constructing the path $D \xrightarrow{*} A.r$ alone proves D is in role $A.r$. However, where $D \xrightarrow{*} A.r$ contains derived edges, the paths they are derived from must be constructed first. The portion of the credential graph that must be constructed is what we call a *credential chain*: $\text{chain}(D \xrightarrow{*} A.r)$ is the least set of edges in E_C containing $D \xrightarrow{*} A.r$ and also containing all the paths that the derived edges in the set are derived from.

3.3 The Backward Search Algorithm

The backward search algorithm determines the member set of a given role expression e_0 . In terms of the credential graph, it finds all the entity nodes that can reach the node e_0 . We call it backward because it follows edges in the reverse direction. This name is consistent with the terminology in X.509 [5, 9], in which forward means going from subjects to issuers and reverse means from issuers to subjects. This algorithm works by constructing *proof graphs*, which are equivalent to, but slightly different from, subgraphs of a credential graph. The minor difference is discussed after the presentation of the algorithm.

The backward search algorithm constructs a proof graph, maintaining a queue of nodes to be processed; both initially contain just one node, e_0 . Nodes are processed one by one until the queue is empty.

To process a role node $A.r$, the algorithm finds all credentials that define $A.r$. For each credential $A.r \leftarrow e$, it creates a node for e , if none exists, and adds the edge $e \rightarrow A.r$. In the proof graph, there is only one node corresponding to each role expression and each edge is added only once. Each time the algorithm tries to create a node for a role expression e , it first checks whether such a node already exists; if not, it creates a new node, adds it into the queue, and returns it. Otherwise, it returns the existing node.

On each node e , the algorithm stores a children set, which is a set of nodes, e_1 , that e can reach directly (i.e., $e \rightarrow e_1$), and a solution set, which is the set of entity nodes, D , that can reach e (i.e., $D \xrightarrow{*} e$). Solutions are propagated from e

to e 's children in the following ways. When a node is notified to add a solution, it checks whether the solution exists in its solution set; if not, it adds the solution and then notifies all its children about this new solution. When a node e_1 is first added as a child of e_2 (as the result of adding $e_2 \rightarrow e_1$), all existing solutions on e_2 are copied to e_1 .

To process an entity node, the algorithm notifies the node to add itself to its own solution set.

To process a linked role node $A.r_1.r_2$, the algorithm creates a node for $A.r_1$ and creates a *linking monitor* to observe the node. The monitor, on observing that $A.r_1$ has received a new solution B , creates a node for $B.r_2$ and adds the edge $B.r_2 \rightarrow A.r_1.r_2$, which we call a link-containment edge.

To process an intersection node $e = f_1 \cap \dots \cap f_k$, the algorithm creates one *intersection monitor*, for e , and k nodes, one for each f_j , then makes the monitor observe each node f_j . This monitor counts how many times it observes that an entity D is added. For a given entity D , each f_j notifies e at most once. If the count reaches k , then the monitor adds the edge $D \rightarrow e$. So, to summarize, in addition to the nodes and edges in the credential graph, the algorithm constructs monitors that implement closure properties 2 and 3.

Given a set of credentials C , the proof graph, $G_{b(e_0, C)}$, constructed by the backward search algorithm starting from e_0 , is closely related to the credential graph, G_C . $G_{b(e_0, C)}$ is almost identical to the smallest subgraph of G_C whose node set, N_C^0 , satisfies the following four closure properties and whose edge set consists of all edges of E_C over nodes of N_C^0 : (i) $e_0 \in N_C^0$; (ii) $e_2 \in N_C^0$ & $e_1 \rightarrow e_2 \in E_C \implies e_1 \in N_C^0$; (iii) $A.r_1.r_2 \in N_C^0 \implies A.r_1 \in N_C^0$; and (iv) $f_1 \cap \dots \cap f_k \in N_C^0$ & $j \in [1..k] \implies f_j \in N_C^0$. The only difference between $G_{b(e_0, C)}$ and such a subgraph of G_C is this: $G_{b(e_0, C)}$ contains role nodes, created during the processing of linked roles, that don't appear in C . Specifically, when the algorithm processes a linked-role node $A.r_1.r_2$, the node $B.r_2$ and the link-containment edge, $B.r_2 \rightarrow A.r_1.r_2$, are added, even when $B.r_2$ does not appear in C , and will therefore receive no incoming edges and no solutions. It is not difficult to see that $G_{b(e_0, C)}$ contains $\text{chain}(D \xrightarrow{*} e_0)$ for every D that can reach e_0 .

THEOREM 3. *Given a set of credentials C , let N be the number of credentials in C , and M be the total size of C : $\sum_{A.r \leftarrow e \in C} |e|$, where $|A| = |A.r| = |A.r_1.r_2| = 1$ and $|f_1 \cap \dots \cap f_k| = k$. Assuming that finding all credentials that define a role takes time linear in the number of such credentials (e.g., by using hashing), then the worst-case time complexity of the backward search algorithm is $O(N^3 + NM)$, and the space complexity is $O(NM)$. If each intersection in C has size $O(N)$, then the time complexity is $O(N^3)$.*

To see that $O(N^3)$ is a tight bound for the algorithm, consider the following example:

$C = \{A_0.r_0 \leftarrow A_i, A_0.r_i \leftarrow A_0.r_{i-1 \bmod n}, A_i.r_0 \leftarrow A_{i-1 \bmod n}.r_0, A_0.r' \leftarrow A_0.r_i.r_0 \mid 0 \leq i < n\}$
There are $N = 4n$ credentials. When using backward search algorithm from $A_0.r'$, there are edges from each $A_j.r_0$ to each $A_0.r_i.r_0$, where $0 \leq i, j < n$, so there are n^2 such edges. Each $A_j.r_0$ gets n solutions, so the time complexity is n^3 . We can see that intersections do not increase the worst-case time complexity of this algorithm. $O(NM)$ is a tight space bound. Following is an example that reaches the bound: $C = \{A_0.r_0 \leftarrow A_i, A_0.r_i \leftarrow A_0.r_{i-1 \bmod n}, A_0.r' \leftarrow A_0.r_i.r_0 \cap A_0.r_i.r_1 \cap \dots \cap A_0.r_i.r_{K-1} \mid 0 \leq i < n\}$

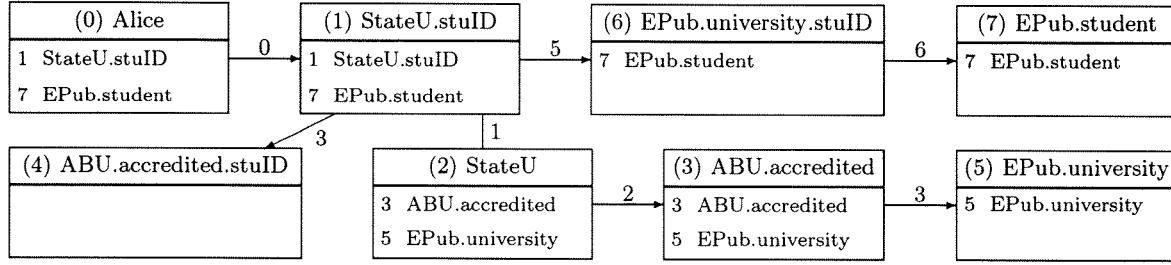


Figure 1: $G_f(\text{Alice}, C)$, the proof graph constructed by doing forward search from Alice with $C = \{\text{EPub.student} \leftarrow \text{EPub.university.student}, \text{EPub.university} \leftarrow \text{ABU.accredited}, \text{ABU.accredited} \leftarrow \text{StateU}, \text{StateU.stuID} \leftarrow \text{Alice}\}$. The first line of each node gives the node number in order of creation and the role expression represented by the node. The second part of a node lists each solution eventually associated with this node. Each of those solutions and each graph edge is labeled by the number of the node that was being processed when the solution or edge was added. The edge labeled with 1 is a linking monitor.

3.4 The Forward Search Algorithm

The forward search algorithm finds all roles that an entity is a member of. The direction of the search moves from the subject of a credential towards its issuer.

The forward algorithm has the same overall structure as the backward algorithm; however, there are some differences. First, each node stores its parents instead of its children. Second, each node e stores two kinds of solutions: full solutions and partial solutions. Each *full solution* on e is a role that e is a member of, *i.e.*, a role node that is reachable from e . Each *partial solution* has the form $(f_1 \cap \dots \cap f_k, j)$, where $1 \leq j \leq k$. The node e gets the solution $(f_1 \cap \dots \cap f_k, j)$ when f_j is reachable from e . Such a partial solution is just one piece of a proof that e can reach $f_1 \cap \dots \cap f_k$. It is passed through edges in the same way as is a full solution. When an entity node D gets the partial solution, it checks whether it has all k pieces; if it does, it creates a node for $f_1 \cap \dots \cap f_k$, if none exists, and adds the edge $D \rightarrow f_1 \cap \dots \cap f_k$.

The processing of each node is also different from that in the backward algorithm. For any role expression e , forward processing involves the following three steps. First, if e is a role $B.r_2$, add itself as a solution to itself, then add a linking monitor observing B . This monitor, when B gets a full solution $A.r_1$, creates the node $A.r_1.r_2$ and adds the edge $B.r_2 \rightarrow A.r_1.r_2$. The addition of such an edge results in $B.r_2$ being added as a parent of $A.r_1.r_2$. Second, find all credentials of the form $A.r \leftarrow e$; for each such credential, create a node for $A.r$, if none exists, and add the edge $e \rightarrow A.r$. Third, if e is not an intersection, find all credentials of the form $A.r \leftarrow f_1 \cap \dots \cap f_k$ such that some $f_j = e$; then add $(f_1 \cap \dots \cap f_k, j)$ as a partial solution on e .

Figure 1 shows the result of doing forward search using a subset of the credentials in example 3.

THEOREM 4. *Under the same assumptions as in theorem 3, the time complexity for the forward search algorithm is $O(N^2M)$, and the space complexity is $O(NM)$.*

3.5 Bi-direction Search Algorithms

When answering queries about whether a given entity, D , is a member of a given role, $A.r$, we have the flexibility of combining forward and backward algorithms into a search that proceeds from both D and $A.r$ at once. In this bi-directional algorithm, a node e stores both its parents and its children, as well as both backward solutions (entities that

are members of e) and forward solutions (roles that e is a member of).

In the centralized case, doing either forward search from D or backward search from $A.r$ suffices to answer the query. However, using bi-directional search could improve search efficiency (where search space size is sometimes exponential in path length) by finding two shorter intersecting paths, rather than one longer one. A variety of search strategies bear consideration, and different algorithms can be developed based on them. The algorithms described above use queues to organize node processing, resulting in breadth-first search. If they used stacks, they would perform depth-first search. In general, when there are several nodes that can be explored (from either direction), they can be placed in a priority queue according to some heuristic criteria, *e.g.*, fan-out. Note that these remarks also apply to the forward and backward algorithms.

In the distributed case, the ability to locate credentials can become a limiting factor. This is the main issue we address in section 4.

3.6 Implementation, Generalization, and Application to SDSI

We have implemented the above algorithms in Java. Our program can be configured to store the parent or child node from which each solution arrives. Using this information, one can easily trace paths, and compute the set of credentials being used in any proof graph.

Our algorithms can be generalized to search for paths between two arbitrary role expressions. One way to do this is to generalize the solution set to collect all reachable nodes, not just entity and role nodes. Then, one knows that a path $e_1 \xrightarrow{*} e_2$ exists when e_1 is added as a backward solution on e_2 or when e_2 is added as a forward solution on e_1 . Of course, such a change would affect the algorithm's complexity.

Our algorithms can also be used to do chain discovery in SDSI. To allow their construction in RT_0 , long linked names can be broken up. Instead of using $A.r \leftarrow B.r_1.r_2 \dots r_k$, one can use $\{A.r \leftarrow A.r'_{k-1}.r_k, A.r'_{k-1} \leftarrow A.r'_{k-2}.r_{k-1}, \dots, A.r'_2 \leftarrow A.r'_1.r_2, A.r'_1 \leftarrow B.r_1\}$, in which the r'_i 's are newly introduced role names. Then one can use any of the algorithms to do goal-oriented chain discovery.

THEOREM 5. *Given a set of "SDSI" credentials C , which have arbitrarily long linked roles and no intersection, let*

C' be the result of breaking up long linked roles. Then the time complexity of the backward algorithm, applied to C' , is $O(N^3L)$, where N is the number of credentials in C , and L is the length of the longest linked role in C .

This $O(N^3L)$ worst-case complexity is the same as that of the algorithm in Clarke *et al.* [8].

Instead of breaking up long linked names, one can extend our algorithms to handle them directly. It is also not difficult to extend our algorithms to handle SPKI delegation certificates. In particular, it is straightforward to extend our techniques for handling intersections to handle threshold structures as well.

4. DISTRIBUTED CHAIN DISCOVERY

The algorithms given in the previous section can be used when credential storage is not centralized, but distributed among credentials' subjects and issuers. As discussed in section 1, it is impractical to require either that all credentials be stored by their issuers or that all be stored by their subjects. Yet if no constraint is imposed on where credentials are stored, some chains cannot be found without broadcast, which we assume is unavailable.

EXAMPLE 4. Consider the following credentials from example 3: $ABU.accredited \leftarrow StateU$ and $StateU.stuID \leftarrow Alice$. If both of these are stored exclusively with $StateU$, none of our search procedures can find the chain that authorizes Alice. Arriving at ABU and at Alice, the procedure is unable to locate either of these two key credentials.

This section presents a type system for credential storage that ensures chains of well-typed credentials can be found.

4.1 Traversability

We introduce notions of path traversability to formalize the three different directions in which distributed chains can be located and assembled, depending on the storage characteristics of their constituent credentials. We call the three notions, *forward traversability*, *backward traversability*, and *confluence*, respectively. Working from one end or the other, or from both simultaneously, a search agent needs to be able to find the credential defining each edge in a path, $D \xrightarrow{*} A.r$, as well as in the other paths of $chain(D \xrightarrow{*} A.r)$, which prove the existence of derived edges in $D \xrightarrow{*} A.r$.

Suppose that $D \xrightarrow{*} A.r$ consists entirely of edges that represent credentials that are stored by their subjects. (In this case, $(D \xrightarrow{*} A.r) = chain(D \xrightarrow{*} A.r)$.) We call $D \xrightarrow{*} A.r$ *forward traversable* because forward search can drive its distributed discovery, as follows. Obtain from D the first credential of the path and, with it, the identity (and hence the location) of the issuer of that credential. That issuer is the subject of the next credential. By visiting each successive entity in the path and requesting their credentials, each credential in the path can be obtained, without broadcast.

A *backward traversable* path is analogous to a forward traversable path, except the credentials involved are held by issuers. A path $D \xrightarrow{*} A.r$ that is backward traversable can be discovered by doing backward search starting from $A.r$. Credentials involved in the path can be collected from entities starting with A and working from issuers to subjects.

Roughly speaking, a *confluent* path can be decomposed into two subpaths, one forward traversable and the other

backward traversable. When both ends are known, a confluent path can be collected and assembled by starting at both ends and working inwards.

We define these notions of traversability for both edges and paths in credential graphs. Following the definition, we discuss the intuition behind traversability of derived edges.

DEFINITION 2 (TRAVERSABILITY AND CONFLUENCE). Let $G_c = \langle N_c, E_c \rangle$ be the credential graph for a given set of credentials, C .

An edge added by closure property 1 is:

- Forward traversable if the credential it represents is held by each subject of the credential;
- Backward traversable if the credential it represents is held by the issuer of the credential;
- Confluent if it is forward or backward traversable.

A path $e_1 \xrightarrow{*} e_2$ is:

- Forward traversable if it is empty ($e_1 = e_2$), or it consists entirely of forward traversable edges;
- Backward traversable if it is empty, or it consists entirely of backward traversable edges;
- Confluent if it is empty, or it can be decomposed into $e_1 \xrightarrow{*} e' \rightarrow e'' \xrightarrow{*} e_2$ where $e_1 \xrightarrow{*} e'$ is forward traversable, $e'' \xrightarrow{*} e_2$ is backward traversable, and $e' \rightarrow e''$ is confluent. Note that paths that are forward traversable or backward traversable are also confluent.

An edge added by closure property 2, $B.r_2 \rightarrow A.r_1.r_2$ is:

- Forward traversable if the path it is derived from, $B \xrightarrow{*} A.r_1$, is forward traversable;
- Backward traversable if $B \xrightarrow{*} A.r_1$ is backward traversable;
- Confluent if $B \xrightarrow{*} A.r_1$ is confluent;

An edge added by closure property 3, $D \rightarrow f_1 \cap \dots \cap f_k$ is :

- Forward traversable if (a) there exists an $\ell \in [1..k]$ with $D \xrightarrow{*} f_\ell$ forward traversable, and (b) for each $j \in [1..k]$, $D \xrightarrow{*} f_j$ is confluent;
- Backward traversable if (a) there exists an $\ell \in [1..k]$ with $D \xrightarrow{*} f_j$ backward traversable, and (b) for each $j \in [1..k]$, $D \xrightarrow{*} f_j$ is confluent;
- Confluent if for each $j \in [1..k]$, $D \xrightarrow{*} f_j$ is confluent;

Here is why a derived edge of the form $B.r_2 \rightarrow A.r_1.r_2$ has the same traversability as the path that it is derived from. Suppose there is a forward traversable path $D \xrightarrow{*} B.r_2 \rightarrow A.r_1.r_2 \xrightarrow{*} A.r$. Starting at D , a search agent can traverse to $B.r_2$. From there, the agent knows B , which enables it to continue searching, traversing $B \xrightarrow{*} A.r_1$. Upon reaching $A.r_1$, the search agent has proven the existence of $B.r_2 \rightarrow A.r_1.r_2$. Additionally, it knows A , so it can continue forward search from $A.r_1.r_2$.

Now suppose there is a forward traversable path $D \xrightarrow{*} A.r$ that can be decomposed into $D \rightarrow f_1 \cap \dots \cap f_k \rightarrow B.r_1 \xrightarrow{*} A.r$. The edge $f_1 \cap \dots \cap f_k \rightarrow B.r_1$ is forward traversable, so it is stored by the entity $base(f_j)$, for each $j \in [1..k]$. If there is one f_ℓ with $D \xrightarrow{*} f_\ell$ forward traversable, a search agent can use it to get from D to f_ℓ . From $base(f_\ell)$, the agent can obtain the credential $B.r_1 \leftarrow f_1 \cap \dots \cap f_k$, thereby identifying all other f_j 's. The search agent then finds a path from

D to each f_j , and continues its forward search from $B.r_1$. Since both ends are known, each path $D \xrightarrow{*} f_j$ only needs to be confluent. The rationale for backward traversability of edges derived from backward traversible paths is similar.

4.2 A Credential Type System

If all credentials are stored by their issuers, all paths are backward traversable. Similarly, if all credentials are stored by their subjects, all paths are forward traversable. As we argued in section 1, neither arrangement by itself is satisfactory—greater flexibility is required in practice. Yet some constraints must be imposed on credential storage, or else many paths cannot be discovered. One way to organize those constraints is by requiring that all credentials defining a given role name have the same storage characteristics. Capitalizing on this observation to support distributed discovery, we introduce a type system for credential storage, the important feature of which is that, given a set of well-typed credentials, every path in its credential graph is confluent.

In our type system, each role name has two types: an issuer-side type specifies whether a search agent can trace credentials that define the role name by starting from the credentials' issuers; the other, a subject-side type, specifies these credentials' traceability from their subjects.

The possible issuer-side type values are *issuer-traces-none*, *issuer-traces-def*, and *issuer-traces-all*. If a role name r is *issuer-traces-def*, then from any entity A one can find all credentials defining $A.r$. In other words, A must store all credentials defining $A.r$. However, this does not guarantee that one can find all members of $A.r$. For instance, we might have $A.r \leftarrow B.r_1$, with r_1 *issuer-traces-none*. This motivates the stronger type: *issuer-traces-all*. A role name r being *issuer-traces-all* implies not only that r is *issuer-traces-def*, but also that, for any entity A , using backward searching, one can find all the members of the role $A.r$.

The possible subject-side type values are *subject-traces-none* and *subject-traces-all*. If a role name r is *subject-traces-all*, then for any entity B , by using forward search, one can find all roles $A.r$ such that B is a member of $A.r$.

There are three values for the issuer-side type and two values for the subject-side type, yielding six combinations; however, a role name that is both *issuer-traces-none* and *subject-traces-none* is useless, so it is forbidden. This is captured by the notion of well-typedness.

We now extend this type system to role expressions and then define the notion of well-typed credentials. As we show in the next section, together these two definitions guarantee that when credentials are well-typed, the following three conditions hold. If a role expression e is *issuer-traces-all*, one can find all members of e by doing backward search from e . If e is *subject-traces-all*, then from any of its members, D , one can find a chain to e by doing forward search. If e is *issuer-traces-def*, then from any of its members, D , one can find a chain from D to e by doing bi-directional search.

DEFINITION 3 (TYPES OF ROLE EXPRESSIONS).

- A role expression is well-typed if it is not both *issuer-traces-none* and *subject-traces-none*.
- An entity A is both *issuer-traces-all* and *subject-traces-all*.
- A role $A.r$ has the same type as r .

- A linked role $A.r_1.r_2$ is

<i>issuer-traces-all</i>	when both r_1 and r_2 are <i>issuer-traces-all</i>
<i>issuer-traces-def</i>	when r_1 is <i>issuer-traces-all</i> and r_2 is <i>issuer-traces-def</i> , or r_1 is <i>issuer-traces-def</i> and r_2 is <i>subject-traces-all</i>
<i>issuer-traces-none</i>	otherwise

<i>subject-traces-all</i>	when both r_1 and r_2 are <i>subject-traces-all</i>
<i>subject-traces-none</i>	otherwise

- An intersection $f_1 \cap \dots \cap f_k$ is

<i>issuer-traces-all</i>	when there exists an f_ℓ that is <i>issuer-traces-all</i> , and all f_j 's are well-typed
<i>issuer-traces-def</i>	when all f_j 's are well-typed
<i>issuer-traces-none</i>	otherwise

<i>subject-traces-all</i>	when there exists an f_ℓ that is <i>subject-traces-all</i> , and all f_j 's are well-typed
<i>subject-traces-none</i>	otherwise

The typing rule for a linked role $A.r_1.r_2$ may need some explanation. If both r_1 and r_2 are *issuer-traces-all*, then from $A.r_1.r_2$, one can find all members of $A.r_1$, and then, for each such member, B , find all members of $B.r_2$. If both r_1 and r_2 are *subject-traces-all*, then from any member, D , of $A.r_1.r_2$, one can first find that D is a member of $B.r_2$, and then find that B is a member of $A.r_1$, thereby determining that D is a member of $A.r_1.r_2$. Knowing both ends, D and $A.r_1.r_2$, one needs to find a middle point, $B.r_2$, using forward or backward search from one side. Then the other side can be handled by bi-direction search. If r_1 is *issuer-traces-all*, one can find all members of $A.r_1$, then r_2 only needs to be *issuer-traces-def*. Similarly, if r_2 is *subject-traces-all*, then one can trace to $B.r_2$ from D , and so r_1 only needs to be *issuer-traces-def*.

DEFINITION 4 (WELL-TYPED CREDENTIALS). A credential $A.r \leftarrow e$ is well-typed if all of the following conditions are satisfied:

1. Both $A.r$ and e are well typed.
2. If $A.r$ is *issuer-traces-all*, e must be *issuer-traces-all*.
3. If $A.r$ is *subject-traces-all*, e must be *subject-traces-all*.
4. If $A.r$ is *issuer-traces-def* or *issuer-traces-all*, A stores this credential.
5. If $A.r$ is *subject-traces-all*, every subject of this credential stores this credential.

Consider credentials in example 3. One possible typing that makes all credentials well-typed is as follows: preferred, spdiscount, student, and university are *issuer-traces-def*, while accredited, stuID, and member are *subject-traces-all*.

4.3 Traversability with Well-typed Credentials

In this section we show that well-typed credentials whose storage is distributed can be located as needed to perform chain discovery.

LEMMA 6. Assume C is a set of well-typed credentials and $G_C = \langle N_C, E_C \rangle$ is the credential graph for C . Let e be any role expression and D any entity. If there is a path $D \xrightarrow{*} e$ in G_C , then we have the following:

1. $D \xrightarrow{*} e$ is confluent.
2. If e is issuer-traces-all, $D \xrightarrow{*} e$ is backward traversable.
3. If e is subject-traces-all, $D \xrightarrow{*} e$ is forward traversable.

From Lemma 6 and Theorems 1 and 2, we have the following theorem, which says that if credentials are well typed, then role membership queries can be solved efficiently, even when credential storage is distributed. This is because confluent paths support efficient chain discovery, as discussed above in section 4.1. Furthermore, for roles of type issuer-traces-all, all members can be found efficiently. Finally, from any entity, it is possibly to find efficiently all subject-traces-all roles to which the entity belongs.

THEOREM 7. Assume that C is a set of well-typed credentials and that $G_C = \langle N_C, E_C \rangle$ is the credential graph for C . Let $A.r$ be any role and B any entity. Then we have the following:

1. $B \in S_C(A.r)$ if and only if there exists a confluent path $B \xrightarrow{*} A.r$ in G_C .
2. If $A.r$ is issuer-traces-all, then $B \in S_C(A.r)$ if and only if there exists a backward traversable path $B \xrightarrow{*} A.r$ in G_C .
3. If $A.r$ is subject-traces-all, then $B \in S_C(A.r)$ if and only if there exists a forward traversable path $B \xrightarrow{*} A.r$ in G_C .

Using a typing scheme such as the one presented here can also help improve the efficiency of centralized search, where type information can help choose nodes to be explored next.

4.4 Agreeing on Types and Role Meanings

Our type system begs the following question: How can entities agree on the type of a role name? This is the problem of establishing a common ontology (vocabulary), and it arises for RT_0 whether or not typing is introduced. Consider again the credentials in example 3. Given $\text{StateU.stuID} \leftarrow \text{Alice}$, how does EPub know what StateU means by stuID ? Is it issued to students registered in any class, or only to students enrolled in a degree program. This issue arises in all trust-management systems. Different entities need a common ontology before they can use each others' credentials. However, name agreement is particularly critical in systems, like RT_0 , that support linked roles. For instance, the expression $\text{EOrg.university.stuID}$ only makes sense when universities use stuID for the same purpose.

We achieve name agreement through a scheme inspired by XML namespaces [7]. One creates what we call *application domain specification documents* (ADSD), defining a suite of related role names. An ADSD gives the types of the role names it defines, as well as natural-language explanations of these role names, including the conditions under which credentials defining these role names should be issued. Credentials contain a preamble in which namespace identifiers are defined to refer to a particular ADSD, e.g., by giving its URI. Each use of a role name inside the credential then

incorporates such a namespace identifier as a prefix. Thus, a relatively short role name specifies a globally unique role name. Each ADSD defines a namespace. Note that this is a different level of namespaces from the notion of namespaces in SDSI. The latter concerns itself with who has the authority to define the members of a role; the former is about establishing common understandings of role names.

5. FUTURE AND RELATED WORK

In this section, we illustrate briefly the next step in our role-based trust-management language work. We then discuss other future directions and related work.

As mentioned in section 2, RT_0 is the first step in a series of role-based trust-management languages. We are extending the algorithms presented here to RT_1 , where role names are terms with internal structure, including logical variables (whose notation starts with "?", as in $?file$). For example, the credential $\text{OS.fileop(delete, ?file)} \leftarrow \text{OS.owner(?file)}$ can be used to express the policy that the operating system will let a file's owner delete the file. We are also working on defining an XML representation for RT_1 credentials and application domain specification documents, as we discussed in section 4.4. RT_1 will be reported in a forthcoming paper.

5.1 Typing and Complete Information

Inferencing based on distributed credentials is often limited by not knowing whether all relevant credentials are present. The standard solution to this problem is to limit the system to monotonic inference rules. This approach ensures that, even without access to all credentials, if the credentials that are present indicate D is a member of $A.r$, it is certainly true. Missing credentials could make you unable to prove D is a member of $A.r$, but cannot lead you to conclude D is a member of $A.r$ erroneously.

When credentials are well-typed, as defined here, this restriction to monotonic inference rules could be relaxed. The type system ensures we know who to contact to request the relevant credentials. So assuming they respond and we trust that they give us the credentials we ask for, we can assume that we obtain all the credentials that are relevant. In this context, it may be safe to use non-monotonic inference rules. This would allow, for instance, granting role membership contingent on not already being a member of another role. This could form a basis for supporting RBAC-style separation of duties, as well as negation as failure. It will be necessary to manage the trust issue. For instance, we may trust that some issuers will give us all relevant credentials, while not trusting some subjects to do the same.

5.2 Credential Sensitivity

Like most prior trust-management work, we assume here that credentials are freely available to the agent responsible for making access control decisions. In general, credentials may themselves be sensitive resources. Techniques have been introduced [18] that support credential exchange in a context where trust management is applied to credentials, as well as to more typical resources. (See [19] for additional references.) That work assumes that credential storage is centralized in two locations: with the resource requester and with the access mediator. It remains open to manage disclosure of sensitive credentials whose storage is distributed among the credential issuers and subjects.

5.3 Other Related Work

In section 2.1, we compared RT_0 credentials with name definition certificates in SDSI 2.0. In section 3.1 we reviewed existing work to chain discovery in SDSI. Now, we discuss some other related work.

QCM (Query Certificate Managers) [12] and QCM's variation SD3 [13] also address distributed credential discovery. The approach in QCM and SD3 assumes that issuer stores all credentials and every query is answered by doing backward searching. As we discussed in the introduction, this is impractical for many applications, including the one illustrated in example 3. Using backward search to determine whether Alice should get the discount requires one to begin by finding all ACM members and all university students.

Graph-based approaches to chain discovery have been used before, *e.g.*, by Aura [1] for SPKI delegation certificates and by Clarke *et al.* [8] for SDSI name certificates without linked names. Neither of them deals with linked names.

6. CONCLUSIONS

We have introduced a role-based trust-management language RT_0 and a set-theoretic semantics for it. We have also introduced credential graphs as a searchable representation of credentials in RT_0 and have proven that reachability in credential graphs is sound and complete with respect to the semantics of RT_0 . Based on credential graphs, we have given goal-oriented algorithms to do credential chain discovery in RT_0 . Because RT_0 is more expressive than SDSI, our algorithms can be used for chain discovery in SDSI, where existing algorithms in the literature either are not goal-oriented or require using specialized logic programming inferencing engines. Because our algorithms are goal-oriented, they can be used whether or not credentials are stored centrally. We have also introduced a type system for credential storage that guarantees distributed, well-typed credential chains can be discovered. This typing approach can be used for other trust-management systems as well.

7. ACKNOWLEDGEMENT

This work is supported by DARPA through AFRL/IF contract F30602-97-C-0336 and SPAWAR contracts N66001-00-C-8015 and N66001-01-C-8005. Sameer Ajmani made some helpful comments on an earlier version of this paper. We also thank anonymous reviewers for their helpful reports.

8. REFERENCES

- [1] Tuomas Aura. Fast Access Control Decisions from Delegation Certificate Databases. In *Proceedings of 3rd Australasian Conference on Information Security and Privacy (ACISP '98)*, volume 1438 of *Lecture Note in Computer Science*, pages 284–295. Springer, 1998.
- [2] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote Trust-Management System, Version 2. IETF RFC 2704, September 1999.
- [3] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, 1996.
- [4] Matt Blaze, Joan Feigenbaum, and Martin Strauss. Compliance-Checking in the PolicyMaker Trust Management System. In *Proceedings of Second International Conference on Financial Cryptography (FC'98)*, volume 1465 of *Lecture Note in Computer Science*, pages 254–274. Springer, 1998.
- [5] Sharon Boeyen, Tim Howes, and Patrick Richard. Internet X.509 Public Key Infrastructure LDAPc2 Schema. IETF RFC 2587, June 1999.
- [6] Piero Bonatti and Pierangela Samarati. Regulating service access and information release on the web. In *Proceedings of the 7th ACM Computer and Communication Security*, pages 134–143, 2000.
- [7] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. W3C Recommendation, January 1999. <http://www.w3.org/TR/REC-xml-names/>.
- [8] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest. Certificate Chain Discovery in SPKI/SDSI. Manuscript submitted to *Journal of Computer Security*, December 2000. Available from <http://theory.lcs.mit.edu/~rivest/publications.html>.
- [9] Yassir Elley, Anne Anderson, Steve Hanna, Sean Mullan, Radia Perlman, and Seth Proctor. Building Certificate Paths: Forward vs. Reverse. In *Proceedings of the 2001 Network and Distributed System Security Symposium (NDSS'01)*, pages 153–160. Internet Society, 2001.
- [10] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI Certificate Theory. IETF RFC 2693, September 1999.
- [11] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. Simple Public Key Certificates. Internet Draft (Work in Progress), July 1999. <http://world.std.com/~cme/spki.txt>.
- [12] Carl A. Gunter and Trevor Jim. Policy-directed certificate retrieval. *Software: Practice & Experience*, 30(15):1609–1640, September 2000.
- [13] Trevor Jim. SD3: a trust management system with certificate evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 106–115. IEEE Computer Society Press, 2001.
- [14] Ninghui Li. Local Names in SPKI/SDSI. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW-13)*, pages 2–15. IEEE Computer Society Press, 2000.
- [15] Ninghui Li, Benjamin N. Grosz, and Joan Feigenbaum. A Practically Implementable and Tractable Delegation Logic. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 27–42. IEEE Computer Society Press, 2000.
- [16] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, February 1996.
- [17] David S. Warren and *et al.* The XSB Programming System (Version 2.2), April 2000. <http://www.cs.sunysb.edu/~sbprolog/xsb-page.html>.
- [18] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated Trust Negotiation. In *DARPA Information Survivability Conference and Exposition*. IEEE Press, January 2000.
- [19] T. Yu, M. Winslett, and K. E. Seamons. Interoperable strategies in automated trust negotiation. In *ACM Conference on Computer and Communications Security*, 2001.

APPENDIX B

Making Trust Negotiation Realistic: A Suitable Policy Language and the Management of Information about Credential Possession

William H. Winsborough and Deborah Shands

Abstract

When principals seek authorization in an attribute-based access control system, the system verifies their attributes by authenticating credentials that document those attributes. Because many attributes are sensitive, principals must protect their credentials. Several strategies have been presented in the literature for the incremental exchange of protected credentials in trust negotiation. However, with the exception of the eager strategy, all strategies presented so far fail to adequately protect information about which credentials the negotiators hold, protecting instead only the full disclosure of the credential. Many existing strategies explicitly acknowledge possession of a credential to anyone who asks. Additionally, prior strategies have used unrealistic authorization policy languages. We present the suitability of the policy language, Delegation Logic, and our adaptation of it for use in ABAC. We also provide a preliminary analysis of issues surrounding the control of access to knowledge of credential possession and content.

1 Introduction

Access control presents difficult problems in a distributed computing environment where resources and the subjects requesting those resources are distributed. Many commonly used access control mechanisms rely heavily on obtaining an authenticated identity of the resource requestor. Unfortunately, when the resource owner and the requestor are relative strangers, access control based on principal identity may be ineffective. In addition, identity-based access control (IBAC) does not scale well in many collaborative environments. In coalition-based collaborative environments, for example, organizations have resource-sharing agreements, but the individual identities of users within those organizations are not necessarily known to coalition partners. The need for a resource owner to be aware of all users from all organizations that might access its resource is a significant administrative burden entailed by the use of IBAC.

Attribute-based access control (ABAC) offers a flexible, scalable alternative to IBAC. Under ABAC, authenticated principal attributes are used to determine whether a subject (acting on behalf of the principal) is permitted to access the requested resource. For example, in the physical world, a university ID card asserts that its subject has “student” status at the issuing university. Many university service providers may request to see the ID card as proof of student status before offering resources to the cardholder. ABAC policies can easily express access conditions that range from tightly restrictive to broadly permissive (e.g., “department chairs only,” or “any current student.”) This offers the flexibility necessary for arbitrarily fine-grained control over access to protected resources.

ABAC offers a highly scalable approach to access control in distributed environments by supporting delegation of attribute authority. By relying upon attribute values issued by other system entities, an access mediator delegates authority over those attributes. For example, when a theater offers a student discount on movie tickets, it often accepts a university ID as evidence of student status. In doing so, the theater delegates authority over student status determination to universities. Delegation of authority over subject principal attributes is critical to the scalability of distributed access control (e.g., the movie theater cannot reasonably maintain student records on its own!)

Scalable access control, supported through delegation of attribute authority, is especially critical in collaborative computing environments. For example, suppose that company DEF supplies component parts that are used in products produced by company ABC. To expedite the supply process, the two companies agree to a purchase order and invoicing system: ABC will authorize certain employees to sign purchase orders; DEF will deliver the parts, based on those orders; and ABC will pay on DEF's invoices for those parts. Under an IBAC system, DEF must know the identity of each ABC employee who is authorized to sign purchase orders. If one of those employees leaves the ABC company or additional purchasing agents are hired, DEF must be informed immediately. Under an ABAC system, the ABC comptroller issues credentials to certain employees, indicating that they have "purchasing authority." DEF's access control system need only recognize and authenticate the "purchasing authority" attribute on receiving a request from an ABC employee. By enabling the ABC comptroller to determine which ABC employees have the "purchasing authority" attribute, DEF reduces its administrative burden significantly. In a distributed computing environment, the scalability of access control systems is heavily dependent on the scalability of administrative processes. ABAC supports scalable access control by enabling scalable security administration based on delegation of attribute authority.

Note that the use of ABAC does not preclude the capture, authentication, and logging of identity information to ensure individual user accountability. ABAC simply removes dependence on principal identity for access decision making. ABAC is also compatible with the use of role-based access control (RBAC) and other access control models, as principal attributes may be used to place the requesting subject into a specific role, according to system policy. ABAC may be used in either a mandatory or a discretionary access control system.

ABAC systems depend on the exchange of credentials which specify principal attributes and/or rules for deriving assertions about principals. Principal attributes (such as financial or medical data) and sometimes derivation rules (such as proprietary actuarial formulas) may be sensitive. The process of *trust negotiation* enables resource requestors and access mediators to establish trust in one another through the cautious, iterative, disclosure of credentials. To support trust negotiation, access policies are established to regulate the disclosure of credentials, as well as the disclosure of information system resources. A variety of trust negotiation strategies and protocols have been developed to facilitate trust establishment. A trust negotiation strategy is an abstract protocol, which defines issues such as when to transmit credentials and, in some cases, policy content, and how to determine that a negotiation has failed. For instance, existing trust negotiation strategies are distinguished by the manner in which they select credentials for disclosure: one strategy, called the eager strategy, exchanges only credentials, while all others exchange credential access policies, which are used in various ways to focus credential disclosures on those relevant to the negotiation.

This paper addresses a critical trust negotiation problem for ABAC. Current trust negotiation strategies do not protect credential possession. Suppose, for example, that Alice holds a government security clearance credential. In her trust negotiation with Bob, she describes conditions under

which she will be willing to disclose that credential. Unfortunately, by providing Bob with her access policy for that credential, Alice has inadvertently revealed that she **holds** a security clearance, though she has not told Bob the contents of that credential (i.e., her level of clearance). We expect that many credentials with sensitive content will also be sensitive to acknowledge possession of. Toward the eventual development of a strategy that addresses this problem, we present here an analysis of this problem of protecting credential possession.

Another deficiency of prior negotiation strategies is that they are based on unrealistic policy languages. Most use a language based on propositional logic, which is entirely unable to meet the needs of ABAC. There is no way to identify acceptable credential issuers by the attributes of those issuers. The most realistic language employed [8] to date is the Trust Policy Language [1]. However, it has the serious flaw that policy rules cannot be contained in credentials. This means that an access mediator is unable to make use flexible use of rules that others have defined for assigning new attributes by interpreting existing attributes. Such power is essential to flexible distribution of authority and for mapping between nomenclatures of different issuers and organizations.

The remainder of the paper is organized as follows. Section 2 provides background on trust negotiation, including a brief summary of five prior strategies. Section 3 presents the problem of protecting information about which credentials a negotiator holds. It also outlines our basic approach to protecting this information in future negotiation strategy designs. Section 4 discusses policy languages for use in trust negotiation and ABAC. It presents the shortcomings of languages used in prior work of this kind, and illustrates our choice of a more suitable language. Section 5 presents a preliminary analysis of safety issues concerning information flow and other characteristics of negotiation strategies. It is aimed at supporting and possibly improving the design approach discussed in Section 3. Section 6 concludes.

2 Background in Trust Negotiation

A trust negotiation consists of sequence of disclosures of credentials (which document attributes) and, in some negotiation strategies, disclosures of policy content that serves to focus the credential disclosures. The principal aim of any negotiation is to satisfy a particular target trust requirement, which in turn enables some desired transaction to take place. A *target trust requirement* (or just a *trust target*) is a policy whose origin may be, for instance, the access policy of a requested target resource. A scenario in which a client process requests access to a server resource is often used to motivate or illustrate negotiation strategies. However, alternative trust target origins may be equally appropriate. For instance, a potential requester may ask a service provider to satisfy a given policy before the requester is willing to reveal sensitive information contained in the planned request. In general, the arrangement is symmetric: each negotiation participant establishes a trust requirement of the other, and the ensuing negotiation seeks to satisfy both. For simplicity of presentation, the remainder of this paper considers the common scenario in which each negotiation attempts to satisfy the trust target set by the access mediator of a service provider.

2.1 Requirements, Expectation, and Assumptions of Trust Negotiation Strategies

All existing trust negotiation strategies meet a short, common list of requirements. In particular, a strategy must:

1. Protect the credential itself. The credential itself provides the potentially sensitive attribute and its field values. It can be used to document this data, not only to the negotiation opponent, but to third parties as well. Thus, the credential must be protected by an access control policy whose satisfaction by a negotiation opponent *unlocks* the credential, signifying that it is safe to transmit it.
2. Terminate. Negotiations must, of course, terminate on either success or failure. Each participant must be able to detect, at some point in the negotiation, whether further exchanges will be futile.
3. Be efficient. Common measures of efficiency for distributed computing protocols are applicable to trust negotiation (e.g., number of messages exchanged, message size). In addition, we may want to quantify local computational complexity of algorithms, for instance, that check whether a specific set of credentials satisfies a given trust target.
4. Be complete. A trust negotiation strategy must not allow a potentially successful negotiation to terminate in failure, assuming both parties are negotiation in good faith, according to the strategy.

We make the following assumptions about credentials, negotiators and policies:

1. There are no universally trusted third parties. We cannot expect that both participants in a trust negotiation will agree to submit all of their credentials and policies to an escrow agent which could then evaluate all the material and render a decision on whether the requester met the trust target. Participants must establish trust through bi-lateral negotiations.
2. Each participant must negotiate in good faith. While every trust negotiation system should protect credential content from attackers (i.e., prevent a negotiator from being tricked into disclosing credentials without prior satisfaction of their access policies), an attacker can easily mislead a negotiator about what to expect in return for credential disclosures. In particular, a service provider may transmit a request for credentials, claiming it comprises the access policy of some resource, but then refuse to provide the resource upon receiving the requested credentials. Satisfaction of this requirement is essential to ensure completeness of any trust negotiation strategy.
3. A participant holds (and is responsible for disclosing) all credentials used to meet trust targets on its behalf. For example, Alice holds both her student ID credential, and a credential from an accreditation body stating that Alice's university is accredited. A movie theater may offer Alice a student discount on her ticket after checking first, that an entity claims that Alice is a student, and, second, that the entity making the claim is a university. The requirement that Alice hold both of these credentials helps to simplify the protocol and completeness arguments. In future work, we plan to explore the consequences of relaxing this assumption.
4. A participant's collection of credentials and policies do not change in the midst of a negotiation. This requirement helps us to simplify the protocol and completeness arguments. If new credentials or policies must be accommodated, a negotiator should terminate the current negotiation and begin a new one.

2.2 Summary of Prior Trust Negotiation Strategies

Several strategies for negotiating trust have been presented in prior work. In many of these strategies, the participants focus the credential exchange by informing one another about which credentials would, if transmitted, further the negotiation. In this section, we sketch the behavior of five strategies, focusing on the material that participants share in order to guide their negotiation.

2.2.1 The *Eager* Strategy

The eager strategy [8] is the simplest trust negotiation strategy and one in which participants share no information which might help to focus the negotiation. Once a trust target is established, each participant sends all of its currently unlocked credentials. Initially, only the unprotected credentials are unlocked. As credentials are exchanged, however, additional credentials become unlocked and are then sent. Eventually, either the trust target is satisfied (i.e., the negotiation succeeds), or the negotiation reaches a point after which no new credentials are unlocked and the trust requirement remains unsatisfied (i.e., the negotiation fails).

Using the eager strategy, negotiators give each other no information about which credentials would be helpful toward meeting the trust target. Consequently, to ensure that negotiations are successful as often as possible, negotiators send every unlocked credential, in case it might be necessary to success. Negotiators also send unlocked credentials as soon as they are unlocked, to minimize the number of credential exchanges. Delay can be introduced without losing completeness, provided negotiators eventually send every unlocked credential unless and until the negotiation succeeds.

2.2.2 The *Parsimonious* Strategy

The parsimonious strategy [8] is one by which participants exchange as much information as possible to focus their negotiation before they begin to exchange credentials. No credentials are exchanged before the participants determine that the trust target can be satisfied. The strategy was named for its stinginess with credential disclosures.

In this strategy, the trust target is the first in a sequence of credential requests exchanged by participants, during an initial phase of the negotiation. Throughout this request exchange phase, the participants, say, Alice and Bob, take turns asking one another for credentials. Each credential request is based on its predecessor. For example, when Alice receives a request from Bob, asking to see the credit limit on her credit card, Alice may reply that she will share that information with Bob only after he has shown her that he represents a legitimate mortgage lender. If Bob satisfies Alice's request, then the access policies on her credit limit information must be satisfied and she may then disclose her credential to Bob.

When either participant, say Bob, receives a request that can be satisfied by transmitting credentials that are unlocked (for instance, because they are not sensitive), the initial phase of the negotiation ends and credentials start to flow. In this case, Bob sends the credentials that solve the last request, while replaying the last request he made to Alice. Because of how Alice's last request was constructed, the set of credentials Bob transmits with his replayed request unlocks a set of Alice's credentials that satisfies Bob's replayed request. Next, Alice transmits the unlocked solution to Bob's replayed request, while also replaying the request she had originally made just prior to the one Bob just replayed. Again, the credentials Alice transmits unlock a solution to

the replayed request that she transmits with them. This process is repeated, running backwards through the sequence of requests, satisfying each of them, until Alice and Bob eventually satisfy the trust target.

Negotiation failure is detected if the initial phase goes on too long. If the negotiation can succeed, it will do so within a number of steps that is less than 4 times the number of credentials either side has.

Negotiators using the parsimonious strategy give each other such precise information about which credentials they need to receive that no credentials flow unless the negotiation will be successful. As we will see in section 3.2, under many circumstances, this turns out to be too much information.

2.2.3 The *Backtracking* Strategy

The backtracking strategy of [9] uses a propositional language, and works by negotiators exchanging requests for specific (propositional) credentials. It is not clear whether it could be extended to a realistic language. However, doing so is unlikely to be fruitful because negotiations require too many messages to flow. The strategy manages to reduce the message number from cubic in the number of credentials held to quadratic by introducing a clever intelligent backtracking technique. However, every other strategy requires only a linear number of messages. The authors attempt to justify their quadratic complexity by claiming (erroneously) that the size of messages in the parsimonious strategy grows exponentially. However, we believe that a backtracking approach cannot be made efficient.

2.2.4 The *All-Relevant Credentials* Strategy

The all-relevant credentials strategy and the all-relevant policies strategies both [7] use the notion of a credential being “syntactically relevant” to satisfying an access policy. In [7], where a propositional language is used, a credential (which is, in that presentation, represented in policies by a propositional variable) is “syntactically relevant” to a policy if it appears there.

Negotiators using the all-relevant-credentials strategy identify to one another credentials that are relevant to satisfying access policies of the target resource and of other relevant credentials. When credentials that are unlocked become relevant, they are transmitted. Eventually, either the trust target is satisfied, or additional credentials neither become relevant nor are transmitted.

2.2.5 The *All-Relevant-Policies* Strategy

Negotiators using the all-relevant-policies strategy exchange policies that govern access to relevant credentials and to other resources. This enables participants to limit transmission of credentials to groups of credentials that satisfy a potentially relevant policy, allowing fewer credentials to flow than in either the eager or the all-relevant credentials strategies. Completeness of the strategy requires that all potentially relevant policies flow. Negotiation failure is detected when no further policies become relevant, neither negotiator can transmit unlocked credentials to satisfy a policy it has received, and yet the trust target remains unsatisfied.

According to the all-relevant-policies strategy, each negotiator transmits access policies for each credential that is “syntactically relevant” to solving one of the policies it received from the other negotiator. The trust target is the first policy transmitted.

3 Possession Protection

This section explains why it is that prior strategies do not adequately protect information about which credentials a negotiator holds. It then explains our plans for protecting this information in future designs. Before either of these, however, we need to introduce the notion of a credential's type.

3.1 Credential Contents versus Credential Type

Our approach to protecting credential possession is based on a notion of credential typing. The type of a credential is typically intuitive. It might be a diploma or a driver's license. More abstractly, a credential's type is attribute that the credential documents. The type does not include the other data contained in the credential, namely attribute field values (parameters).

The difference between content protection and possession protection depends on where in the credential structure the information of interest is represented. If possession of a credential is acknowledged, we mean that possession of a credential having a certain type is acknowledged. On the other hand, when we talk about credential content, we are talking about attribute fields or parameters. For example, a driver's license might be a credential's type, while age, hair color, and class of vehicle would be fields.

Sometimes it is clear whether information belongs in a credential's type or in its parameters. Some information is clearly content: where parameters have values, like age, the values are clearly content. But elsewhere the boundary is vague, possibly even arbitrary. Consider a credential documenting membership in a state's legal bar association. In principle, the type could just be "membership." However, it might also be appropriate to identify the organization in the credential's type. We do not attempt to prescribe in this situation. Rather, we simply point out that information contained in a credential's type will be unprotected unless possession protection measures are adopted.

3.2 Possession Protection in Prior Strategies

Among the five strategies outlined above in Section 2.2, none but the eager strategy [8] successfully protects which credentials a negotiator has. The eager strategy discloses no information of any kind about a credential until that credential's access control policy has been satisfied. This is because no policy content is exchanged at all. In effect, this means that no credential is even mentioned by either party until it is unlocked. In every other strategy, the response to a credential request can disclose whether the requested credentials are held.

The backtracking strategy [9] discloses which credentials are possessed when the credentials are requested. The credentials treated in that work are propositional, in the sense that they contain no parameters. All the information content that affects policy evaluation, and hence authorization decisions, is disclosed freely to anyone who asks. The only content to which access is controlled is the digital signature that enables the credential to be verified. While the propositional nature of credentials is not essential to Ting's backtracking strategy, what *is* essential is the requirement that, without prior trust establishment, negotiators must respond to queries about whether they have credentials that satisfy specific policy requirements provided in the query.

The all-relevant-credentials and all-relevant-policies strategies [7] provide access control policies (or some simplified information about them) for credentials when anyone requests those credentials.

Since this information is provided only for credentials actually held, it can be used by anyone to determine what credentials a negotiator holds.

In the parsimonious strategy [8], negotiators are required to respond to a request for credentials by presenting a counter request for opponent credentials that are logically necessary and sufficient to unlock a set of credentials that satisfies the original request. This means, for example, that if Alice receives a request from Bob for a credential showing she has top-secret clearance, Alice has to respond in a way that discloses to Bob whether or not she has such a credential. Specifically, if Alice has the credential, she has to respond by transmitting the access policy that governs it, and if she do not, she has to respond by indicating that the negotiation cannot succeed. So while the strategy does not actually transmit the credential itself unless doing so is authorized, it discloses information about which credentials Alice has in an uncontrolled way.

We believe that negotiators will have too many credentials for the eager strategy to provide adequate performance. Some amount of policy information will be exchanged in order to focus credential disclosures on relevant credentials. Among the existing strategies that do this, we believe that the all-relevant-policies strategy is the best suited for adaptation to possession protection. In particular, the basic design of both the parsimonious and the backtracking strategies seem to be incompatible with possession protection. The all-relevant-credentials strategy, which has a similar outline, could be a candidate as well, though we expect it to be far less effective than the all-relevant-policies strategy in limiting irrelevant credential transmissions.

In the next section we outline our approach to adapting the all-relevant-policies strategy to protect credential possession.

3.3 Our Basic Approach to Possession Protection

The outline of the negotiation strategy we envision is as follows. This outline is based on the all-relevant-policies strategy [7]. It ignores obvious optimizations, like avoiding retransmission of credentials sent in previous messages.

1. Alice's Strategy. Takes: (BobCreds, BobRequests)
2. Verify signatures on BobCreds.
3. If Bob is the resource requester, check whether BobCreds unlock access to requested resource. If so, negotiation succeeds.
4. Determine the set of Alice's credentials that are relevant to satisfying BobRequests.
5. Determine Alice's requests for Bob's credentials.
6. If that set has changed since the last time, send them to Bob.
7. Otherwise,
 - (a) Check for failure, and
 - (b) Find a set of credentials to disclose to Bob.

Our basic approach to protecting information about which credentials are possessed is this. In step 4, Alice determines which credential types are relevant to satisfying Bob’s credential request. In step 5, for each relevant credential type, whether she holds such a credential or not, Alice always transmits a policy that governs acknowledging credentials of that type. Once Bob satisfies this *acknowledgement policy* (which we also sometimes call *possession protection policy*), Alice transmits access control policies (disclosure policies) for each credential she holds of that type. (If there are none, that fact can be inferred because no access control policies are transmitted.)

To facilitate negotiators being able to transmit acknowledgement policies for credentials they do not hold, we suggest that issuers post suggested acknowledgement policies for each type of credential that they issue. Inevitably, a negotiator will not possess acknowledgement policies for every type of credential that they do not possess. However, this just signifies that the negotiator does not consider non-possession of that credential to be sensitive. Note that negotiators are motivated to obtain some acknowledgement policies for credentials they do not have, even if they don’t consider that fact sensitive, just because doing so reduces the information content of providing an acknowledgement policy.

4 Language and Definitions

This section outlines the suitability and adaptation of Delegation Logic (DL) [4] for use in trust negotiation and ABAC. It begins by discussing the shortcomings of languages used in prior trust negotiation work. It then illustrates the use of key DL language features by considering an ABAC example. It goes on to present our notion of credential type. It concludes by formalizing how DL is used to make access control decisions.

4.1 Prior Languages Used in Trust Negotiation

Several languages besides DL have been discussed as candidates for policy specification. Here we discuss the shortcomings of the alternative languages, motivating our selection of DL. The key distinguishing feature turns out to require a special approach for managing possession protection. The section concludes with a summary of language requirements.

In [6], Prolog was used to specify access policies. Credentials contained purely extensional information: facts, but no rules for deriving new facts. The policy served to map those facts to more highly abstract attributes, called roles in [6]. Prolog proved quite flexible and expressive. However, Prolog is unsatisfactory as a production policy language because it is excessively expressive, flexible, and low-level. A production policy language should be special purpose, abstracting out issues that are common to all policies.

Propositional Logic, used in [9] and [7] does not explicitly designate issuer or subject of credentials. It is a central function of attribute-based access policy to specify trusted issuers. Consequently, purely propositional languages can serve only illustrative purposes. Additionally, it is necessary to link the subject of one credential with the issuer of another to be able to delegate attribute authority based on issuer attributes.

The Trust Policy Language (TPL) [1] is an attribute-based policy language that maps credentials to groups, which the authors propose be used as a basis for access control. Among the alternatives to DL, TPL is the best for ABAC. However, there are two missing features. One important omission is that TPL attributes (called interchangeably “groups” and “roles”) are atomic, and cannot have

parameters, which significantly inhibits policy modularity. The second omission is more critical, however: TPL does not allow credentials to specify rules that can be used to derive attributes from other attributes. This means that all rules for interpreting credentials in any given authorization decision must be written by the same author.

The inability to use rules written by others (without manual administrative action) is a serious limitation. Rules written by other can translate attributes from one nomenclature to another. They can be used to interpret credentials issued by third parties. They provide enormous flexibility.

Thus, the feature of DL that most distinguishes its suitability for our purpose from the best of its competitors is DL’s ability to express rules, issued on the authority of someone other than the access policy author, that can be used to derive new attributes. In other words, attributes can be defined intensionally as well as extensionally. Rules can be encapsulated for transmission in verifiable credentials, just as more conventional extensional attributes are. We call credentials that contain rules *intensional credentials*.

This feature of DL—support for intensional credentials—has a interesting and important impact on trust negotiation. Rules can be used to prove a principal has an attribute, but only if additional supporting attributes can first be proven. Now trust negotiation must enable the transmission of credentials that prove those supporting attributes, as well as transmission of the rule. In strategies that transmit access policy content, it must be transmitted for the additional supporting credentials. As we see below in section 5.2.1, the timing of the transmission of this additional policy material is key to reliable possession protection.

We conclude this section with a summary of ABAC language requirements:

1. Clear, high-level semantics. General purpose languages are not appropriate because they overburden policy and credential authors. Procedural semantics and, worse, semantics based on a reference implementation are inappropriate because their complexity will introduce ambiguity and variation among policy interpretations.
2. Monotonic. Particularly because negotiators control the availability of their own credentials, it would be unacceptable to confer more authorizations when a credential is withheld.
3. Parametric attributes. Field values, such as age or spending limit, must be representable.
4. Ability to delegate attribute authority to entities based on their own attributes. This is a key to ABAC’s scalability.
5. Ability to use attribute-derivation rules defined by others. Such rules need to be allowed in credentials, where their authenticity can be verified.

4.2 Using Delegation Logic for ABAC

To enable ABAC, access policies must be specified in an expressive language with unambiguous semantics. Unfortunately, the implementation-dependencies of an operational semantics can be particularly problematic in a distributed environment, where we are most interested in using ABAC. Delegation Logic (DL) [4] overcomes this issue by providing a model-theoretic semantics of policy compliance. Additionally, DL provides rich expressive power for the assessment of trustworthiness, the conferment of trust, and the specification of rules for reasoning about trustworthiness and trust

decisions. In this section, we introduce portions of Delegation Logic (DL) that we propose to use in trust negotiation.

Intuitively, DL enables the formal expression of rules for determining when credentials are sufficient to satisfy a policy. A brief discussion of common implementations of a few example DL clauses may help to provide an intuitive understanding of their meaning before we proceed with a discussion of the formalisms. Consider the following DL clause, from the purchasing example of Section 1:

HR says job_title(Alice, ‘‘purchasing_agent’’). (1)

In clause (1), “HR” and “Alice” stand for specific principals, represented by their public cryptographic keys. “HR” represents the human resources department of the ABC company, while Alice is an ABC employee. “job_title(Alice, ‘‘purchasing_agent’’)” means that the value of Alice’s “job_title” attribute is “purchasing_agent.” Clause (1) can be embedded in a credential, signed by the HR department. This allows the credential recipient to verify the integrity and authenticity of the clause, which says that the HR department asserts that Alice is a purchasing agent.

Continuing the example, the ABC company comptroller may want to allow purchasing agents to sign purchase orders (up to \$5000) for any of the corporate partners with whom ABC has a purchasing agreement. By using the job title assertion from the HR department, the ABC comptroller delegates authority over Alice’s job title to the HR department:

ABC says purchase_authority(Alice, 5000) if (2)
HR says job_title(Alice, ‘‘purchasing_agent’’).

The comptroller can use the assertion of clause (1), together with its own rule (2) to derive the assessment:

ABC says purchase_authority(Alice, 5000). (3)

Of course, the comptroller would not invent a new purchase authority rule for each ABC employee, but will instead use a more general rule:

ABC says purchase_authority(?X, 5000) if (4)
HR says job_title(?X, ‘‘purchasing_agent’’).

In DL, specific principals are denoted by *constants* (e.g., “ABC,” “Alice,” and “HR”). Logical variables, such as ?X, denote unspecified principals. A DL *variable*, which begins with question mark, can be systematically replaced throughout a clause by an arbitrary logical term—in this case, by any constant representing a principal. If Alice is substituted for ?X in (4), we obtain (2).

DL provides constructs which can be used to express various forms of delegation of authority, a concept central to ABAC. In using (4), the ABC comptroller delegated authority over employee job titles to the HR department. More sophisticated forms of delegation are necessary, however, to support scalability for ABAC. One such form is the ability to delegate credential issuing authority, based on the delegatee’s attributes. For example,

DEF says sell(?X, Widget) if (5)
?Y says purchase_authority(?X, ?Z) AND
DEF says business_partner(?Y) AND
?Z ≥ 1000.

The rule expressed in (5) says that the DEF company will honor a Widget purchase request signed by an individual who is granted purchase authority by a business partner. The spending limit on the purchase authority attribute must be at least \$1000, to accomodate the price of Widgets. This example illustrates the advantage of delegating authority, based on the delegatee’s attributes: the policy does not explicitly list all of the companies that might assign purchase authority. Rather, it “generically” delegates purchase authority to any company that has the “`business_partner`” attribute. This method of delegation is critical to two aspects of administrative scalability. First, it enables a modestly-sized ABAC policy to use the expertise of a broad base of credential authorities without specifically mentioning each credential authority that might eventually be employed. Trust assessment rules need not necessarily grow in number, complexity, or specificity, as the user population or number of services grows. Second, the policy administrator need not **know** every credential authority whose credentials might be used to satisfy the policy. This reduces the scope of knowledge and administrative experience required of the policy administrator. By enabling attribute-based delegation of credential issuing authority, DL supports both of these important aspects of scalability for ABAC administration.

Rules (4) and (5) can be combined with the following two facts to allow DEF to conclude that Bob may purchase a Widget:

HR says job_title(Bob, ‘‘purchasing_agent’’)

(6)

DEF says business_partner(ABC)

(7)

To determine whether Bob’s Widget purchase order should be accepted, we make the following query:

DEF says sell(?Bob, Widget)?

Now that we have an intuitive understanding of some simple DL clauses, we can define some terminology that will be used throughout the remainder of the paper. In DL, **says** and **if** are reserved words. The left-hand side of the **if** is called the *head* of the clause (e.g., from (4), “ABC says purchase_authority(?X, 5000)”). The right-hand side is called the *body* of the clause (e.g., from (4), “HR says job_title(?X, ‘‘purchasing_agent’’)”). The **if** expresses a right-to-left implication. A DL clause whose body is empty asserts the head unconditionally. Such a clause is written without the **if** and is called a *fact* (e.g., from (7), “DEF says business_partner(ABC)”). A clause where the body is not empty is called a *rule* (e.g., (5)). The *issuer* of “DEF says business_partner(ABC)” is DEF. The issuer of a clause head is also the issuer of the clause.

The DL clauses that we discuss in this paper are composed of direct statements. A *direct statement* takes the form Princ says Pred($t_1, \dots, t_n$) (e.g., from (6), “HR says job_title(Bob, ‘‘purchasing_agent’’)”). Here, Princ is a principal denoted by a variable or a constant, Pred is a *predicate symbol* (e.g., “business_partner” from (7)), and $t_1, \dots, t_n$ are logical terms. A *logical term* is a constant, a variable, or a *compound term*—a function symbol applied to a tuple of other logical terms. Clauses (5) and (7) above use the logical term “business_partner” to represent an attribute/value pair. Note that a function application is not evaluable in logic programming, but is rather a kind of tree data structure. Strictly speaking, DL does not permit function symbols to take arguments, allowing only 0-arity functions (i.e., constants). This is known as the “datalog” restriction, which is logically sufficient to ensure that evaluation of ordinary logic programs is

tractable. However, this restriction can be relaxed to allow bounded depth compound terms without jeopardizing tractability.

The semantics of DL clauses is derived from the semantics of ordinary logic programs (OLP). For example,

$$(4), (5), (6), (7) \models \text{DEF says sell}(\text{Bob}, \text{Widget}) \quad (8)$$

where the left-hand side of the $\models$ consists of the four clauses, (4), (5), (6), (7). By $\models$, we mean classical logical entailment, which states that every model of the left-hand side is a model of the right-hand side. On both sides, the DL clauses are (implicitly) mechanically transformed into OLP (see [4]), enabling the semantic foundation of OLP to be applied (see for instance [5]). DL queries can then be evaluated by using the transformation and then applying one of the available efficient implementations of the sound and complete proof procedure for OLP.

The version of DL we use here is called Delegation Logic Programs version 1 (D1LP) in [4, 2]. (This language is slightly different from the earlier language of the same name presented in [3] and was introduced in [4] to correct technical tractability issues with the earlier version.) When we refer to DL in this paper, we mean D1LP as described in [4].

The semantics (and the implementation) of DL programs transform them into *definite* OLP programs, which do not use logical negation. As a consequence, our policies are *monotonic*. This means that, for example, if entailment (8) holds and more clauses are added to the left-hand side, then the right-hand side (“DEF says sell(Bob, Widget)”) continues to follow. For our purposes, monotonicity implies that consideration of newly obtained credentials will not reverse a previous positive determination of policy compliance. Conversely, policy compliance cannot be achieved by virtue of withholding some credentials. We view the monotonicity property as essential in a context where requestors control and protect the credentials that can be used to check policy compliance.

4.3 Credentials and Credential Types

A credential contains a collection of attributes regarding its subject and, thus, is a critical element of an ABAC system. In this section, we use DL to formalize the notion of a credential and develop a notion of credential type. The latter notion has not been precisely specified in prior trust negotiation strategies and is critical both to the precision of the possession protection strategy specification and to the development of the failure detection algorithm.

A *credential* **Cred** represents a set containing one or more DL clauses. The issuer of the clauses in a credential must be the principal that signs the credential. The set of clauses in **Cred** can be recovered by using the function application `ExtractClauses(Cred)`. We also apply `ExtractClauses` to sets of credentials, where it returns the union of the resulting clause sets.

We say that a clause *defines* the predicate used in its head. Thus, rule (5) above defines `sell`. We extend the idea of a clause defining a predicate to that of a credential defining a predicate. We formalize this idea by the relation, *defines*, which we write:

$$\text{Cred defines Pred} \quad \text{iff} \quad \text{Pred is defined by some clause in } \text{ExtractClauses}(\text{Cred}).$$

This suggests a notion of credential type: if **Cred** defines **Pred**, we say that **Cred** has type **Pred**. Since **Cred** defines **Pred** may hold for several values of **Pred**, credentials may have polymorphic type. This is a weak notion of type, in that it does not provide any sense of type safety, but is sufficient for our purposes.

Our principal requirement of a type is that it identify a category of credentials that are relevant to policy content that is transmitted during trust negotiation. This notion of credential type provides an abstraction that serves two important purposes. First, it supports the determination of credential relevance. For example, Alice may request that Bob present a credential of type “driver’s license,” without telling him exactly which fields of the license she intends to examine. This allows her to indicate to Bob which credentials she might accept, without enumerating all of the DMV authorities which may suffice. Second, the notion of credential type permits a negotiator to avoid unnecessary disclosure of field values. In our example, Bob may admit to having a credential of type driver’s license without being forced to also specify that his license is for Bob Smith, fifty years old, brown eyes, brown hair, 150 lbs., glasses required, etc. Alice and Bob can negotiate the types of the credentials that they may disclose to one another, before discussing some of the potentially sensitive details of those credentials. Thus, by introducing the abstraction of credential type, we enable participants to begin the negotiation concentrating on what kinds of credentials they are willing to admit having, and then to continue by negotiating which ones they will actually show each other.

4.4 Access Policy

An ABAC access policy specifies rules, ultimately based on authenticated requestor attributes, for determining whether a protected resource may be released to a requestor. In a trust negotiation, protected resources may include application or system objects (usually, the *target resource*, or primary goal of the access request), as well as credentials, knowledge of the existence of credentials, and policies, themselves. Access policies must, therefore, be applied to each of these entities. When access to a resource R is protected in accordance with a policy $\mathcal{P}$, we refer to $\mathcal{P}$ as the policy that *governs* R . In this section, we provide formal definitions of access policy and a semantics for policy compliance.

We have chosen DL as our policy expression language due to its expressive power and axiomatic semantics (through translation into OLP). Definite Trust Policy Language (DTPL), described in [1], is also a monotonic language which can be mapped to OLP. Though DTPL allows a policy compliance checker to make use of facts provided by another authority, it does not support the use of policy rules developed by other authorities. For example, DTPL would allow the HR department to use a fact such as (7) in evaluating a rule such as (5). It would not, however, permit DEF to use a rule such as (4), signed by the ABC comptroller, to determine whether a given ABC employee has purchase authority. The ability to share such rules for reasoning is critical to scalable access control in a distributed system.

An *access policy* (or simply a *policy*) is given by a set $\mathcal{P}$ of DL clauses. For instance, we might have $\mathcal{P}=\{(4), (5)\}$, where (4) and (5) are clauses from above. A DL engine is used to check the sufficiency of requestor credentials against target resource access policy during the authorization process. We assume that issuer identity and credential content integrity are first verified in the usual manner. Clauses contained in requestor credentials are extracted and then combined with the set of clauses that make up the local policy governing the target resource. This combined set of clauses is loaded into a DL evaluation engine, and a query corresponding to the request for the target resource/action is posed to the engine. If the query can be satisfied, then the action is authorized; otherwise it is not. When the target resource/action is the disclosure of a specific credential, then the query we pose to the DL engine uses the “discloseTo” predicate as in:

$$\text{Alice says discloseTo(DEF)?} \tag{9}$$

This frames the task of designing a DL policy: a policy must define the query predicate so that the query is true exactly when authorization is appropriate. For instance, Alice may prefer to release her sensitive “`purchase_authority`” attribute only to entities that have a legitimate need for that information (i.e., ABC’s business partners). The policy governing the release of her “`purchase_authority`” might be a rule such as:

$$\begin{aligned} &\text{Alice says discloseTo(?Y) if} \\ &\quad \text{ABC says business_partner(?Y).} \end{aligned} \tag{10}$$

The DL engine implements a semantics defined in terms of logical entailment. We now use that same semantics to define a Boolean-valued function, `authorize`, which indicates whether a requestor with a given set of credentials complies with a specific access policy. Recall from section 4.3 that, for a given set of credentials, `RequestorCreds`, `ExtractClauses(RequestorCreds)` yields the set of clauses contained in any of the credentials.

Definition 4.1 (`authorize`): For any policy, $\mathcal{P}$, any set of credentials, `RequestorCreds`, and any requestor key, `RequestorKey`, define `authorize` by:

$$\begin{aligned} &\text{authorize}(\mathcal{P}, \text{RequestorCreds}, \text{RequestorKey}) \\ &\quad \equiv \text{ExtractClauses}(\text{RequestorCreds}) \cup \mathcal{P} \models \text{discloseTo}(\text{RequestorKey}). \end{aligned}$$

Here $\models$ denotes classical logical entailment.

5 Analysis

This section presents a preliminary analysis of safety concerns regarding information flow and completeness of trust negotiation. It is aimed at supporting or improving the design ideas outlined in section 3. As the findings presented are highly preliminary, some readers may wish to skim this section.

5.1 Credential Relevance

The strategy algorithm sketched in section 3.3 above uses (in Line 5) the notion of credentials that are relevant to a request. Intuitively, a credential is treated as relevant if it could potentially assist in satisfying an oponent policy governing either the target resource or another relevant credential. Roughly speaking, if one of Alice’s credentials is relevant to a policy she receives (in Line 1) from Bob, then the policies governing that credential must be transmitted to Bob, enabling Bob to try to unlock Alice’s credential. The reason why Bob transmits policies to Alice in the first place is to enable Alice to limit her credential disclosures to credentials that are relevant. However, if Alice’s definition of relevance is too narrow, causing her to overlook important credentials, the negotiation may fail unnecessarily.

This section examines what it means for a credential to be relevant to a negotiation. As we shall see, several notions of relevance are possible, and the notion selected has a variety of impacts on negotiation characteristics. For instance, the fewer credentials that are deemed relevant, the fewer credentials (and policies) actually flow during negotiation, which supports the principal of least privilege.

On the other hand, a credential must be deemed relevant to have its access policy transmitted. This is the defining characteristic shared by different notions of relevance in our discussion: relevant credentials are the ones that have their acknowledgement and access policies transmitted. If the opponent does not receive the access policy, it may not transmit credentials needed to unlock an important opponent credential. Consequently failing to treat enough credentials as relevant can destroy a strategy's completeness.

We begin by making the latter observation precise, establishing a lower bound on the number of credentials that are deemed relevant and on the set of associated credentials that have to be transmitted to support strategy completeness. Then we perform a variety of other amazing feats that will be certain either to win us national acclaim, or to cause us to lose our jobs and be thrown in jail, one or the other.

5.1.1 A Spectrum of Relevance Relations

This section introduces the notion of a relevance relation, which is the concept we use to formalise a credential's relevance to a policy. The simplest type of relevance relation consists of credential/policy pairs where the credential could participate in satisfying the policy. We call this kind of relevance *simple relevance*. As we shall now see, it works well when credentials contain facts, but no rules.

Assuming Credentials Contain Only Facts

When credentials contain only facts, a natural notion of simple relevance is one in which a credential is relevant to a request if it participates in a minimal solution of that request. That is, it is part of a set of credentials that is a solution of the request, and no proper subset of that solution is also a solution. We call this *minimal-solution relevance*.

The cost of determining whether a given credential can participate in a minimal solution to a given policy is likely to be quite high. This holds true, even if a negotiator has a modest number of credentials, since simple relevance is formalized as a single, universal binary relation that does not consider which credentials a negotiator actually possesses. Fortunately, over estimating the set of credentials that is relevant to a policy is probably acceptable. (As we shall see below, the important thing is not to underestimate this set.)

In general, we call one relevance relation *stronger* than another if it is formalized by a smaller set of policy/credential pairs. The strongest kind of simple relevance that supports negotiation completeness is the minimal-solution variety. In principle any relation containing the minimal-solution relation as a subset also can yield a complete negotiation strategy. For illustration, we describe four of the most natural examples.

Trivial relevance. This relation makes every credential relevant to every policy.

Same-predicate relevance. This relation considers a credential relevant to a policy if the credential defines a predicate that is used by the policy (i.e., that occurs in a policy clause body).

Unifiability relevance. Here, a credential is relevant to a policy if it contains a fact that unifies with a call in a policy clause body. Clearly unifiability relevance is a stronger notion of relevance than same-predicate relevance.

Same-predicate, issuer, and subject relevance. This is like unifiability, except that not all arguments are required to unify. Of course, there are several different versions of this notion, depending on which arguments are required to unify. The one named here has the issuer and subject unifiable.

When Credentials Contain Arbitrary Clauses

When credentials can contain rules, minimal-solution simple relevance degenerates to being the same as trivial relevance: *i.e.*, it makes every credential relevant to every policy. (To see this, consider any policy, p_1 , and any credential c_1 that participate in a minimal solution of p_1 . Now consider any credential, c_2 . To see that c_2 also participates in a minimal solution consider a credential that is identical to c_1 except that the head of c_2 is added to c_1 's body.)

Of course the problem is that minimal-solution simple relevance considers all possible credentials, not just the ones actually held by a negotiator. For this we introduce a family of relations, each one with its own credential pool.

For instance, we might write

$$R_{CP} \subseteq \text{Creds} \times \text{Policies}$$

for a relevance relation for credential pool $CP \subseteq \text{Creds}$. In general, we call such relevance relations *pool-specific*. For instance, a pool-specific minimal-solution relation R_{CP} would have $R_{CP}(c, p)$ hold if $c \in C \subseteq CP$ and C is a minimal solution of p .

Note that even in the facts-only case, the pool-specific minimal-solution relevance relation is stronger than the simple minimal-solution relation. The difference between them is that the simple relation considers as relevant credentials that would have to be combined with credentials not currently held in order to satisfy a policy. The pool-specific relation does not.

In the present discussion, we generally assume that credentials held by a requester are the only credentials that will be considered in support of an access request. One can imagine designs where this assumption would have to be relaxed. For instance, some designs allowing credentials to be accumulated by the access mediator might require this. However, without restricting the form of credentials accumulated by the access mediator, all credentials become relevant to all policies, as we have seen. Though part of important future work, these, and other problems that arise if we relax the assumption that credentials are accumulated solely by the requester, are beyond the current scope.

As in the facts-only case above, efficiency concerns necessitate consideration of relevance relations that approximate this strongest, safe notion of pool-specific relevance—the minimal-solution notion. And as above, interesting candidate definitions use predicate matching or unification. Now, with rules in credentials, the definitions must be extended to consider credentials whose clause heads match bodies of other relevant credentials. We call these notions of relevance *deep-matching* relations, to distinguish them from the notions discussed above, which we call *shallow-matching* relations. (Although shallow-matching relations do not safely approximate the pool-specific minimal-solution relation, we shall see that they are useful in organizing incremental disclosures of authorization and access policies. Note that they are easily extended to credentials containing rules. For example, the definition of (shallow) unifiability relevance above says that “a credential is relevant to a policy if it contains a *fact* that unifies with a call in a policy clause body.” In the context of credentials containing rules, this should be read as “a credential is relevant to a policy if it contains a *clause* whose head unifies with a call in a policy clause body.”)

Deep-matching, pool-specific, same-predicate relevance. For a given credential pool, CP , this relation considers a credential $c \in CP$ relevant to p if the clause in c defines a predicate that is used in the body of a clause that is in p or in some other $c' \in CP$ that is also relevant to p .

Deep-matching, pool-specific, unifiability relevance. This relation resembles the one above, except that the head of the clause in c must unify with a call in a clause in p or in some other $c' \in CP$ that is also relevant to p .

Deep-matching, pool-specific, partial unifiability relevance. This relation resembles deep-matching, pool-specific, unifiability relevance, except that not all predicate parameters have to unify.

To a first approximation, all relevant credentials must have their policies transmitted. However, one aspect of relevance that becomes particularly important for controlling information flow when credentials can contain rules is the fact that not every relevant credential has to have its authorization and access policies transmitted immediately. Waiting is controlled by a *policy transmission strategy*. The limits on how long you can wait, as well as a lower bound on which credentials have to be considered relevant, is given in section 5.1.2. In subsequent sections, we study how that freedom is essential to protecting information about credential possession.

5.1.2 Safety Requirement on Credential Relevance

The aim of this section is to clarify the intuition that in strategies based on the outline given in 3.3, credentials are not transmitted unless there is some indication that they are relevant to negotiation success. So far our characterization of relevance is rather imprecise. For one thing, it bears examining what it means for a credential “potentially to assist in satisfying an oponent policy.” Moreover, it is not essential that every policy governing every such credential be transmitted immediately as soon as it can be identified. As mentioned above and explored further in later sections, a variety of notions of relevance are possible. Our immediate goal is to clarify minimal safety requirements of each candidate notion that ensure negotiation is appropriately supported.

We are interested here only in complete strategies. Thus, our safety requirement must ensure that (good-faith) participants provide to one another sufficient information to identify every credential that might need to be transmitted for the negotiation to succeed.

For a given credential, **Cred**, there is a (possibly empty) set of policies that must be satisfied before **Cred** can be transmitted. Let us denote this set by $govPolicies(\mathbf{Cred})$. This set may contain access policies and acknowledgement policies. (If some policies in the set are protected, as discussed in [7], the set will also contain additional policies which must be satisfied before the protected policies are transmitted and subsequently satisfied.) Let us extend $govPolicies$ to sets of credentials by taking the union of the function values on the set elements, so that if **CredSet** is a set of credentials, we have

$$govPolicies(\mathbf{CredSet}) = \bigcup_{\mathbf{Cred} \in \mathbf{CredSet}} govPolicies(\mathbf{Cred}).$$

Now we can state our safety requirement. Note that rather than focusing on the set of relevant credentials, it is in effect a closure property on the set of transmitted policies, *transPolicies*. This

is natural, given that the key characteristic of relevant credentials is that they are the ones that have their acknowledgement and access policies transmitted:

Requirement 5.1 *When a negotiation is declared a failure, for each minimal solution, sol , to a transmitted policy $pol \in transPolicies$, if $govPolicies(sol) \not\subseteq transPolicies$ then there exists a policy in $govPolicies(sol) \cap transPolicies$ that cannot be satisfied.*

This characterization of our safety requirement allows a short-circuit-style, lazy evaluation of logical *and*: if a set of policies would all have to be satisfied to unlock a resource, and one of the policies cannot be satisfied, the others do not have to be evaluated, or even transmitted. We use the resulting flexibility below in section 5.2.1, where we show how delaying transmission of policies governing some relevant credentials enable possession protection for credentials that contain rules.

5.2 Possession Protection and Relevance

When acknowledgement policies are used to support possession protection, the credentials that have their acknowledgement policies transmitted are those that are relevant to requests recieved. This includes credentials that are not held, as well as those that are. So relevance now determines which acknowledgement policies—as well as which access policies—are candidates for transmission, subject to the policy transmission strategy.

5.2.1 Impact of Rules in Credentials for Possession Protection

As we have seen, when credentials contain rules, simple minimal-solution relevance degenerates to trivial relevance. The problem is that for every policy and every credential, there is another hypothetical credential that can be used as part of a solution to the policy just in case the first credential is also used. Also as we have seen, what a simple relevance relation fails to consider is the pool of credentials that the negotiator actually possesses. This can be overcome by using a credential-pool relevance relation. However, this presents certain problems when we aim to protect information about which credentials are possessed: credential-pool relevance relations naturally contain information about the credential pool. Such information is leaked if, for instance, acknowledgement policies are transmitted for credentials whose relevance to a request is indirect, via a second credential in the pool. In particular, the presence of that second credential in the pool is disclosed.

The solution to this problem is to use a shallow relevance relation and to add to the policy the clauses in credentials that are already unlocked. This approach is safe with respect to Requirement 5.1.

5.2.2 Relevance Classes and Acknowledgement Classes

By definition, Alice’s relevant credentials are the ones whose policies should (eventually) be transmitted. So Alice’s relevant credentials are all actually held by Alice. However, for the purpose of analyzing information flow, it is useful to consider the behavior of Alice’s relevance relation over the whole space of possible credentials, not just over the credentials Alice happens to possess.

A given relevance relation defines an equivalence relation over credentials where two credentials are equivalent if, for every policy, they are either both relevant or both irrelevant to the policy.

The space of possible credentials is partitioned into equivalence classes by this equivalence relation. We call these equivalence classes *relevance classes* and the induced partition a *relevance partition*. Different relevance relations induce different relevance partitions. For instance, shallow predicate-match relevance induces relevance classes that consist of credentials that all define the same predicate, and shallow unification relevance induces relevance classes that consist of credentials whose clause heads are all identical, up to variable renaming. The relevance classes that consist of credentials that are relevant to a given policy are called the *relevance classes of that policy*.

Relevance classes are sets of credentials that, subject to timing considerations imposed by an acknowledgement strategy, have their acknowledgement policies and possibly their access policies transmitted in response to the same requests.

A related notion in our approach to possession protection is the partitioning of credentials into sets that are governed by the same acknowledgement policies. We call these classes *acknowledgement classes*. Each negotiator must conceptually partition the entire space of possible credentials into acknowledgement classes and associate policies with each class. In principle, any partition is possible. However, as we shall see, there is an important relationship between the acknowledgement partition and the relevance partition, which leads to a natural preference for certain partitions.

In practice, participants may actually represent only a fraction of the acknowledgement classes they define, depending on the granularity of their acknowledgement partition. They must be prepared to construct new classes on the fly, and to associate acknowledgement policies with them, should elements of those classes be relevant to a request received during negotiation. Consequently, the part of the partition that is explicitly represented will tend to grow over time, as the negotiator defines acknowledgement policies for additional classes—sometimes on the fly.

There is an important relationship between the two partitions defined by relevance classes and by acknowledgement classes. To a great extent, negotiators are free to choose their own definitions of these two partitions. However, the acknowledgement partition should always be either the same as or a refinement of the relevance partition. This means that each relevance class should be the union of one or more acknowledgement classes.

We set aside for the moment the effect of policy transmission strategy, and assume that, once a relevant credential's acknowledgement governing policy is satisfied, its access policy is transmitted. Suppose a negotiator has a relevance class that is not the union of a collection of acknowledgement classes. That means there are credentials that are governed by the same acknowledgement policy and that are relevant to different policies.

Suppose Alice has an acknowledgement class policy A_1 that is governed by *ackPolicy*₁, which Bob satisfies. And suppose that Alice uses a relevance relation such that two of the induced relevance classes, R_1 and R_2 , each intersects A_1 . Bob could now pose a request for R_1 and then a request for R_2 , and, by observing whether he gets back any policies from Alice in each case, determine whether Alice has any credentials in each of the two classes. This is additional information beyond what Bob is entitled to know at this point, which is whether Alice has any credentials in A_1 .

To make this more concrete, let us consider an example. Suppose that Alice associates acknowledgement policies with credentials according to the predicate defined, so, for instance, all credentials defining the predicate, *diploma*, are governed by the same acknowledgement policy. Once Bob satisfies that acknowledgement policy, he is entitled to know that Alice has credentials defining *diploma*. Suppose that Alice has *diploma* credentials:

1. UMichgan says *diploma*(Alice, B.S., 1986, Physics).

2. Purdue says `diploma(Alice, M.S., 1988, ComputerScience)`

Now suppose that Alice uses a relevance relation based on minimal solutions. If Bob sends Alice a request of the form

UniversityX? says `diploma(Alice, degree?, year?, major?)?`

Alice will respond with two policies, p_1 and p_2 , corresponding to the numbered credentials shown above. However, if Bob then sends a request of the form

UniversityX? says `diploma(Alice, Ph.D., year?, major?)?`

Alice will send no policies, and Bob can infer that she doesn't have a Ph.D. credential. This is not authorized information for Bob to receive, however. All Bob is entitled to know, based on satisfying the acknowledgement policy for diploma credentials is that Alice has some diploma credentials. Alice can avoid this potential leakage of information by making her acknowledgement classes smaller or by making her relevance classes bigger. For instance, she could make her relevance relation be based on simply matching predicates. Then when Bob requests any kind of diploma credential, once he has satisfied the acknowledgement policy, he always just receives p_1 and p_2 from Alice. So all he learns is that Alice has at least two diploma credentials, but he doesn't discover anything about their contents.

Alternatively, Alice could plug the leak by making her acknowledgement classes be the same as the minimal-solution relevance classes. This means that each acknowledgement class consists of a single credential. In this case, satisfying the acknowledgement policy means that Bob is authorized to know whether Alice has each specific credential, for instance, whether Alice has a Ph.D. from the University of Toronto from 1992. This is considerably more information than simply knowing Alice has some degree credentials, which may or may not have her as the subject, as was the information authorized by the acknowledgement policy in the original example. Not only would Alice want to make the acknowledgement policy correspondingly strong, but she would need to keep track of potentially many, many more acknowledgement policies. Note that Alice can always use the same policy for as many credentials as she wishes. What changes if she uses the finer minimal-solution-based acknowledgement classes is the amount of information that Bob is authorized to receive if he satisfies the acknowledgement policy. Note that in this case Alice would have essentially no need for credential access policies, since the acknowledgement policy should only be satisfiable by entities that are authorized to know the full contents of the credential.

5.3 Future Analysis Work

Our discussion here has framed certain efficiency issues, but leaves it for future work to follow up on several open problems. Among these are (1) techniques for analyzing policy dependences, (2) policy management support that allows the credentials that are relevant to a negotiation to be pinpointed without unsafe information flow, and (3) other methods designed to shorten negotiations.

6 Conclusion

We have proposed a realistic policy language for use in trust negotiation. We have also analyzed the problem of controlling the flow of information about which credentials are held. Both proposal

and analysis are preliminary and likely to require revision as we continue this line of research, in which we aim to provide a realistic trust negotiation strategy.

References

- [1] Amir Herzberg, Yosi Mass, Joris Mihaeli, Dalit Naor, and Yiftach Ravid. Access control meets public key infrastructure, or: Assigning roles to strangers. In *IEEE 1999 Symposium on Security and Privacy*, Oakland, 1999. IEEE Press.
- [2] Ninghui Li. *Delegation Logic: A Logic-based Approach to Distributed Authorization*. PhD thesis, Department of Computer Science, New York University, September 2000.
- [3] Ninghui Li, Benjamin Grosf, and Joan Feigenbaum. A logic-based knowledge representation for authorization with delegation. Technical Report RC21492, IBM Research, June 1999. Extended abstract appears in *Proceedings of the 12th IEEE Computer Security Foundations Workshop*.
- [4] Ninghui Li, Benjamin Grosf, and Joan Feigenbaum. A practically implementable and tractable delegation logic. In *IEEE 2000 Symposium on Security and Privacy*, Oakland, 2000. IEEE Press.
- [5] John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1984.
- [6] K. Seamons, W. Winsborough, and M. Winslett. Internet credential acceptance policies. In *Proceedings of the 2nd International Workshop on Logic Programming Tools for Internet Applications*, July 1997.
- [7] Kent E. Seamons, Marianne Winslett, and Ting Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Network and Distributed System Security Symposium*. IEEE Press, January 2001.
- [8] William H. Winsborough, Kent E. Seamons, and Vicki E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition*. IEEE Press, January 2000.
- [9] T. Yu, X. Ma, and M. Winslett. Prunes: An efficient and complete strategy for trust negotiation over the internet. In *ACM Conference on Computer and Communications Security*, 2000.

Appendix C

Advances in Trust Negotiation

Final Report

Submitted to NAI Labs

AFRL contract F30602-97-C-0336

December 13, 2000

Principal Investigator

Dr. Kent E. Seamons
Brigham Young University
Computer Science Department
3370 TMCB
Provo, Utah 84602
Phone: (801) 378-3722
Fax: (801) 378-7775
Email: seamons@cs.byu.edu

Summary

The proliferation of Internet-enabled devices will usher in an age of anytime, anywhere computing. When a client and server begin an interaction, most often they will be strangers, with no pre-existing relationship. Mutual trust establishment between them will not be based on identity, but rather on other properties of interest encoded in digital credentials, the online analogues of today's paper credentials. For example, a web server may be interested in providing service only to clients residing in a particular state. Knowing the exact identity of a client (e.g., SSN) is of no help when deciding whether to provide service. Instead, the client needs to give the server a digital credential that attests to their state of residence. Prior to disclosing that credential, the client may require a credential from the server attesting to its proper handling of private information. A bilateral exchange of digital credentials can be used to establish mutual trust gradually, a process called *trust negotiation*. The research during the project focused on the understanding and development of trust negotiation strategies, support for sensitive access control policies, and an architecture for proactive trust establishment for sensitive content.

Trust Negotiation. There are a huge number of possible strategies for negotiating trust, each with different properties with respect to speed of negotiations and caution in giving out credentials and security policies. In the autonomous world of the Internet, entities will want to choose strategies that meet their own goals, which means that two strangers who negotiate trust will rarely use the same strategy. To date, only a tiny fraction of the space of possible strategies has been explored, and no two of the strategies proposed so far will interoperate. During this project, we conducted the first investigation into support for a protocol for trust negotiation that would allow negotiation strategies to interoperate.

Proactive Trust Establishment. A client's initial request to a server may include sensitive information in the body of the request message such as SSNs, credit card numbers, travel destinations, or medical information. Clients will often require assurance that servers are authorized to receive sensitive content and that they will handle it appropriately according to well-established policies that the client or the client's institution establishes or follows. Clients need that assurance in advance of making the actual service request. Thus, they must proactively establish trust based on the content of the request *before* the request is sent to the server. This project examined the requirements of an architecture for proactive trust establishment for sensitive content. The goal is an architecture that scales from individual use to an organization with thousands of clients.

Summary. This research has advanced the understanding of principles surrounding trust negotiation that will empower individuals and organizations with scalable trust establishment approaches that protect privacy and sensitive content to enable ubiquitous trust negotiation.

Deliverables. The deliverables for this project include the remainder of this document that describes an architecture for proactive trust establishment and an attached article, "TrustBuilder: Middleware to Enable Trust Negotiation Strategy Interoperability." The article outlines a protocol to allow two parties to independently select their trust negotiation strategy and have the strategies interoperate. The article is joint work with Professor Marianne Winslett and Ting Yu at the University of Illinois at Urbana-Champaign.

Proactive trust establishment for sensitive content

A client's initial request to a service provider sometimes includes sensitive information such as social security numbers, credit card numbers, travel destinations, monetary amounts, or medical information in the body of the request message. Clients will often require assurance that service providers are authorized to receive sensitive content and that they will handle it appropriately according to well-established policies that the client, client's institution, or the client's coalition establishes. Clients need that assurance in advance of making the actual service request. Thus, they must proactively establish trust based on the content of the request *before* the request is sent to the server.

Trust negotiation capabilities must be expanded to a broader context than in the past in order to handle privacy requirements for sensitive content effectively. Previous work in trust negotiation dealt only with servers initiating trust negotiations in response to a client attempting to access a secure service. During the negotiation, either party could attempt to establish trust in the other party prior to disclosing a credential to them. Thus, clients could establish trust in servers only in the context of credential disclosure during a trust negotiation. Servers could establish trust in the client for other purposes in addition to credential disclosure, but only in response to a request for service, never proactively.

The need for proactive trust establishment for sensitive content goes beyond the context of a web browser contacting an unfamiliar web server to include any process that is pushing sensitive content to an unknown destination. For instance, web servers that push sensitive content to web caches or any point-of-presence on the Internet will have similar trust requirements. An automated solution to proactive trust establishment, rather than a manual solution, will be advantageous for potentially sensitive content that must be distributed rapidly to meet business needs.

Prior advances in trust negotiation offer a partial solution to the problem of proactive trust establishment. For example, languages for expressing policies and negotiation strategy developments can all be re-used in this context. There are two aspects of the problem that earlier work in trust negotiation did not address that will be addressed in this project. The first of these is the problem of determining the policy that specifies the trust requirements associated with sensitive content. Often, this determination will need to be made dynamically at runtime when the content is generated instead of relying on static associations between policy and content made by administrators in advance. The second aspect is the problem of determining the appropriate trust negotiation representative associated with a service that will receive the sensitive content so that trust can be legitimately established prior to initiating the sensitive service request.

1. Dynamic policy association

Prior to anyone accessing a sensitive service and credential, a user or administrator creates an access control policy to protect it. The association between the policy and the secure resource is static. Managing that association at runtime is straightforward. Some sensitive content may not exist until runtime, such as when a web-based purchase order is constructed at the client just prior to invoking the purchasing service. The security agent that must proactively establish trust

may need to dynamically associate a policy with the content at runtime. It cannot rely upon static policy associations established a priori, but must create the association on demand. We call this a *content-based access control policy*.

The determination of the appropriate content-based policy for establishing trust in a server must be under client control. The appropriate policy cannot be statically associated by the server or service and be delivered to the client on demand before a service request is made because different clients may demand different assurances from the same server. For example, with respect to privacy policies, clients in the United States will be interested in knowing that servers adhere to self-regulating guidelines such as TRUSTe. European clients will be interested in following EU privacy directives and should divulge private information only to servers that adhere to the principles of “Safe Harbor” [8].

To allow a client to determine and enforce a content-based access control policy prior to making a sensitive service request, we propose the run-time architecture for dynamic policy association for sensitive content shown in Figure 1. A content analysis engine inspects out-bound content to map it to a content disclosure policy that is enforced by the local security agent before the content can be disclosed.

A content analysis engine must be able to handle many types of data to be useful in practice. In object-oriented systems such as CORBA [20], Enterprise JavaBeans (EJB) [19], or Java, the object model can be leveraged in determining the semantics of content. This will allow content disclosure policies to be associated with application-level objects. The analysis engine can map content to appropriate security policies based on the type or value of the content. Structured data formats, such as XML [22] are suitable for reliable and efficient policy association if the semantics of the data are captured in the structured data representation. For instance, XML data tags and HTTP POST messages can capture semantic information in tags or labels. Other semi-structured data techniques [5] [6] and natural language understanding [15] [16] [18] technology will need to be employed to protect a broader range of content composed of less-structured data types. In time, standard conventions and terminology will develop for highly sensitive data so that appropriate privacy controls can be enforced.

An important design consideration is where to position the content analysis engine in the trust negotiation architecture. CORBA interceptors [20] allow functionality such as security to be

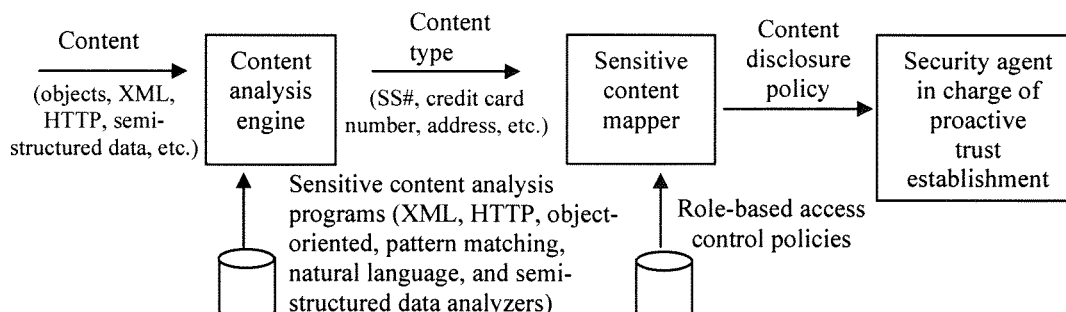


Figure 1. A run-time architecture for dynamic policy association during proactive trust establishment. A content analysis engine accepts out-bound content and determines its type using an arsenal of content analysis techniques. The content type is mapped to a content disclosure policy that is enforced prior to communicating the sensitive content to the intended recipient.

transparently inserted into the dynamic method invocation process. Using interceptor technology, a CORBA Object Request Broker (ORB) that supports dynamic method invocation could be modified to identify sensitive content and establish trust before invoking a remote method with method arguments that include sensitive content. A browser plug-in or a trusted proxy server could intercept client requests and determine if the content is sensitive so that additional trust should be established in the server before forwarding the request. Servlet technology [13] is another vehicle for developing a general-purpose content-analysis engine capable of wide deployment in web architectures. A proxy server is configurable so that it can sit at the edge of the network inside a firewall and provide consistent, mandatory content disclosure policy association for an entire organization. The same functionality could be embedded in an application-level gateway firewall. The proxy server or firewall approach eases administrative overhead, as it does not rely on clients configuring their local environment or require the installation of additional software on each machine. It also permits transparent interception of all outgoing requests, potentially dramatically improving the controls organizations and households have over their sensitive content. However, the overhead of examining each request could be prohibitive. A browser plug-in offers similar functionality, but at the individual browser level. This would allow individual users fine-grained control over their personal content at the cost of more administrative overhead.

One difficult problem is to recognize the meaning of the content and map that meaning to the corresponding policy specification governing disclosure of the content. An approach to solving this problem is as follows. First, evaluate existing publicly available implementations of techniques for determining the meaning of content such as pattern matching, object-oriented type information, semi-structured data, and natural language processing. Second, compare their suitability for inclusion in the content analysis engine in terms of accuracy and speed in determining the meaning of content. The focus of this effort would not be on developing new methods for understanding the semantics of content. It would be useful to create a performance model for content analysis that can be used to predict the costs and overhead for a given set of classification categories and content.

The trust establishment architecture for sensitive content must be highly scalable in terms of administration and runtime performance. To provide administrative scalability, the architecture will leverage a role-based access control model [23]. Roles can be thought of as the intensional analogue to the extensional groups widely used for access control in file systems. The process for protecting sensitive content at runtime proceeds as follows. First, a content-analysis engine maps sensitive content to a role-based authorization policy associated with the content. The policy specifies the roles to which outside parties must authenticate in order to receive the content. Next, the appropriate authentication policy information in terms of credentials is combined with the role expression to be sent to the trust establishment agent that represents the server.

Different administrators, even across organizational boundaries, can manage role definitions in terms of credentials and manage content disclosure policies in terms of roles. This separation of roles and credentials improves administrative scalability. This isolation of credentials within authorization policies also allows our results to be utilized to protect the exchange of sensitive content in systems where the entities are familiar to one another and credentials are not used.

Scalability is also important for managing policy information. Some policies will be shared across many users. Rather than replicate policy data, references to policies could allow policy updates to be immediately reflected in the runtime environment. The separation of policy information into role expressions for authorization and authentication policies in terms of credentials allows the authentication policies to be distributed across different locations and accessed on demand. For instance, a government agency may be responsible for specifying the authentication policy for US citizens and the US Post Office may specify the authentication policy for a US resident. When an authorization policy includes both of these roles, the security agent that is proactively establishing trust could gather the authorization policies on demand in order to submit them to the server. The resulting increase in ease of administration must be weighed against the increased runtime costs. In sum, administrative scalability will be enabled through role-based authorization policies and distributed authentication policies.

2. Determining an appropriate trust negotiation agent

Previous work in trust negotiation did not need to address the issue of determining the trust negotiation agent with which to initiate a negotiation. The server reacted to a client request for service and initiated a trust negotiation with that client. Proactive trust establishment cannot leverage that advantage because the negotiation must take place in advance of the first contact for service. Standard conventions need to be adopted that prescribe the relationship between a service and the associated trust establishment agent representing the service. The solution cannot require clients to disclose, even inadvertently, sensitive information about the service request they intend to submit. There are several ways this could be done using existing technology. Further, solutions to this problem must scale as the number of clients and servers grows. The administrative overhead must not become a detriment to practical deployment. Also, interoperability is important. For example, any CORBA client ought to be able to negotiate with any CORBA server, regardless of the vendor. Timely research in this area will be able to influence future standards.

To find the appropriate trust negotiation agent, object-oriented systems such as CORBA and Java could provide a standard interface where calls could be made prior to invoking a method with sensitive argument values. Objects needing to provide proactive trust establishment could inherit from a proactive trust establishment class and support that interface. If the method name is standardized, client code that dispatches dynamic method invocations could examine the object definitions and determine at run time if the object supports an interface for proactive trust establishment, and invoke it first before invoking a method accepting sensitive content as an argument.

A web server could support trust establishment in advance of a sensitive request in several ways. First, a new HTTP message type could be introduced for the purpose of establishing trust in the web server prior to making a specific request of the server. Second, a new servlet API could be introduced for the purpose of proactive trust establishment. Third, the TLS/SSL protocol [9] [11] could be enhanced to support a more sophisticated version of server authentication that supports trust negotiation rather than the limited client/server authentication it supports today. Fourth, servers can provide a standardized directory service where clients obtain

a reference to the authorized trust establishment agent for a given service. This will allow greater flexibility and local autonomy at the overhead of a directory lookup.

3. References

- [1] E. Bina, V. Jones, R. McCool and M. Winslett, "Secure Access to Data Over the Internet," *ACM/IEEE International Conference on Parallel and Distributed Information Systems*, Austin, Texas, September 1994.
- [2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust Management System Version 2," Internet Draft RFC 2704, September 1999.
- [3] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "KeyNote: Trust Management for Public-Key Infrastructures," *Security Protocols*, 6th International Workshop, Cambridge, UK, 1998.
- [4] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," *IEEE Symposium on Security and Privacy*, Oakland, May 1996.
- [5] P. Buneman, "Semistructured Data," *Proceedings of the Sixteenth ACM SIGMOD Symposium on Principles of Database Systems*, Tucson, Arizona, 1997, pp. 117-121
- [6] P. Buneman, Susan B. Davidson, Mary F. Fernandez, Dan Suciu, "Adding Structure to Unstructured Data," *International Conference on Database Theory*, Delphi, Greece, 1997.
- [7] T. Dierks, C. Allen, "The TLS Protocol Version 1.0," draft-ietf-tls-protocol-06.txt, Nov. 12, 1998.
- [8] The European Union, "Directive 95/46/EC: on the protection of individuals with regard to the processing of personal data and on the free movement of such data." Available at http://www.privacy.org/pi/intl_orgs/ec/eudp.html.
- [9] S. Farrell, "TLS Extensions for Attribute Certificate Based Authorization," draft-ietf-tls-attr-cert-01.txt, Aug. 20, 1998.
- [10] B. Friedman, P. Kahn Jr., and D. Howe, "Trust Online," *Communications of the ACM*, Vol. 43, No. 12, December 2000.
- [11] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol," Netscape Communications Corp., Nov. 18, 1996.
- [12] A. Herzberg, J. 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, May 2000.
- [13] J. Hunter and W. Crawford, "Java Servlet Programming," *O'Reilly & Associates*, 1988.
- [14] N. Islam, R. Anand, T. Jaeger, and J. R. Rao. "A flexible security system for using Internet content." *IEEE Software*, Vol. 14, No. 5, September - October 1997.
- [15] Karen Jensen, George E. Heidorn, and Stephen D. Richardson (editors), "Natural Language Processing: The PLNLP Approach," *Kluwer Academic Publishers*, Boston, 1993.
- [16] M. King, "Evaluating natural language processing systems," *Communications of the ACM* 39:1, January 1996, pp. 73-39.
- [17] W. Johnston, S. Mudumbai, and M. Thompson, "Authorization and Attribute Certificates for Widely Distributed Access Control," *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1998.

- [18] D.D. Lewis and K. Sparck Jones, "Natural Language Processing for Information Retrieval," *Communications of the ACM* 39:1, January 1996, pp. 92-101.
- [19] R. Monson-Haefel, "Enterprise JavaBeans," 2nd edition, O'Reilly & Associates, 2000.
- [20] Object Management Group, "The Common Object Request Broker: Architecture and Specification," Rev. 2.31, October 1999. Available at <http://www.omg.org/technology/documents/formal/corba2formal.htm>.
- [21] "Platform for Privacy Preferences (P3P) Specification," W3C, <http://www.w3.org/TR/WD-P3P/Overview.html>.
- [22] P. Prescod and C. Goldfarb, "The XML Handbook," 2nd edition, *Prentice Hall*, 1999.
- [23] Ravi S. Sandhu and Edward J. Coyne, "Role-Based Access Control Models," *IEEE Computer*, Volume 29, Number 2, February 1996, pages 38-47.
- [24] B. Schneier, *Applied Cryptography*, John Wiley and Sons, Inc., second edition, 1996.
- [25] Simple Public Key Infrastructure (SPKI), <http://www.ietf.org/html.charters/spki-charter.html>.
- [26] W. Winsborough, K. Seamons, and V. Jones, "Negotiating Disclosure of Sensitive Credentials", Second Conference on Security in Communication Networks, Amalfi, Italy, September 1999.
- [27] W. Winsborough, K. Seamons, and V. Jones, "Automated Trust Negotiation," *DARPA Information Survivability Conference and Exposition*, Hilton Head, January 2000.
- [28] W. Winsborough, K. Seamons, and V. Jones, "Automated Trust Negotiation," submitted for journal publication, April 2000, currently available at <http://www.csc.ncsu.edu/faculty/vej/atn.ps>.
- [29] M. Winslett, N. Ching, V. Jones, and I. Slepchin, "Using Digital Credentials on the World-Wide Web," *Journal of Computer Security*, 5, 1997, 255-267.
- [30] International Telecommunication Union, Rec. X.509 - Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, August 1997.
- [31] T. Yu, X. Ma, and M. Winslett, "PRUNES: An Efficient and Complete Strategy for Automated Trust Negotiation over the Internet," *ACM Conference on Computer and Communications Security*, Athens, Greece, November 2000.

Appendix D

FINAL REPORT

For the UIUC Subcontract Under
Advances in Trust Negotiation
An NAI Project Funded by DARPA

July 2001

Principal Investigator

Professor Marianne Winslett
Database Research Laboratory
Department of Computer Science
University of Illinois
1304 West Springfield Avenue
Urbana, IL 61801 USA
winslett@cs.uiuc.edu
(217) 333-3536

Administrative Point of Contact

Patrick M. Patterson
Department of Computer Science
University of Illinois
1304 West Springfield Avenue
Urbana, IL 61801 USA
pmpatter@cs.uiuc.edu
(217) 333-0028

1. The Nature of the Deliverables

The tangible deliverables for this project took the form of written communication with William Winsborough at NAI, plus two documents written by Professor Winslett, her PhD student Ting Yu who was supported by this grant, and coauthors. These documents are listed below. They are available over the web, and are summarized briefly in the sections that follow.

T. Yu, M. Winslett, and K. E. Seamons, "Interoperable Strategies in Automated Trust Negotiation," accepted to the ACM Conference on Computer and Communications Security, 2001. Available on line at <http://drl.cs.uiuc.edu/ccs2001.ps>.

T. Yu, M. Winslett, and K. E. Seamons, extended version of "Interoperable Strategies in Automated Trust Negotiation." This extended version will be the basis for a journal paper submission this fall. Available on line at http://drl.cs.uiuc.edu/nai_report.ps.

2. Research Results

The goal of this subcontract was to address a bothersome aspect of all the trust negotiation strategies proposed to date, including our own proposals: they would not interoperate. In other words, two parties had to use the exact same negotiation strategy, if they were to have any hope of negotiating trust. Our goal was to remove this restriction, by coming up with classes of interoperable trust negotiation strategies. We did this in the paper that has just been accepted to CCS 2001. (The current version of the paper is the submitted version, not the final version (due August 15, 2001). Because CCS 2001 reviewing was double-blind, no author names appear on the submitted version of the paper. The actual authors are Ting Yu, Marianne Winslett, and Kent E. Seamons.) In this paper, we present two large classes of negotiation strategies with the property that negotiation participants can pick any two strategies belonging to the same class, and be assured of correct interoperation. We explore many interesting formal properties of the new strategy classes in this paper, whose abstract is reproduced below.

Automated trust negotiation is an approach to establishing trust between strangers through the exchange of digital credentials and the use of access control policies that specify what combinations of credentials a stranger must disclose in order to gain access to each local service or credential. We introduce the concept of a trust negotiation protocol, which defines the ordering of messages and the type of information messages will contain. To carry out trust negotiation, a party pairs its negotiation protocol with a trust negotiation strategy that controls the exact content of the messages, i.e., which credentials to disclose, when to disclose them, and when to terminate a negotiation. There are a huge number of possible strategies for negotiating trust, each with different properties with respect to speed of negotiations and caution in giving out credentials and policies. In the autonomous world of the Internet, entities will want the freedom to choose negotiation strategies that meet their own goals, which means that two strangers who negotiate trust will often not use the same strategy. To date, only a tiny fraction of the space of possible negotiation strategies has been explored, and no two of the strategies proposed so far will interoperate. In this paper, we define a large set of strategies called the disclosure tree strategy (DTS) family. Then we prove that if two parties each choose strategies from the DTS family, then they will be able to negotiate trust as well as if they were both using the same strategy. Further, they can change strategies at any point during negotiation. We also show that the DTS family is closed, i.e., any strategy that can interoperate with every strategy in the DTS family must also be a member of the DTS family. We

also give examples of practical strategies that belong to the DTS family and fit within the TrustBuilder architecture and protocol for trust negotiation.

The CCS 2001 paper was written for single-level policies; in other words, it did not handle policy graphs, a mechanism that can be used to protect sensitive access-policy content. In fact, it would not be possible to squeeze policy graphs into the CCS 2001 submission, which was already bursting at the seams. The only suitable venue for the extension of the CCS 2001 paper to policy graphs is a journal submission. We have a draft of this submission, obtained by adding several new sections to the end of the paper, extending the definitions and results to apply to policy graphs. Before this paper can be submitted to a journal, it needs three improvements. First, the English in the new sections must be polished, and a new sentence added to the abstract, mentioning the extension to policy graphs. Second, a large and realistic example that uses policy graphs, rather than single-level policies, must be added to the paper. Third, a description of the experimental implementation of the approach needs to be added. This implementation of the TrustBuilder framework for trust negotiation is being done by our colleague Kent Seamons at Brigham Young University, and the implementation is being built this summer. While the paper's focus will remain theoretical, we believe that a journal submission needs to have at least a small practical component. When the paper is ready, we plan to send it to one of the society *Transactions* journals. The work will be completed under a follow-on subcontract through BYU, for a DARPA grant administered by WPAFB; and by another follow-on subcontract through BYU, for a SPAWAR grant.

Appendix E

TrustBuilder: Middleware to Enable Trust Negotiation Strategy Interoperability

ABSTRACT

Automated trust negotiation is a new approach to establishing trust between strangers through the exchange of digital credentials and the use of access control policies that specify what combinations of credentials a stranger must disclose in order to gain access to each local service or credential. We say that a trust negotiation protocol defines the ordering of messages and the type of information messages will contain, and a trust negotiation strategy controls the exact content of the messages, i.e., which credentials to disclose, when to disclose them, and when to terminate a negotiation. There are a huge number of possible strategies for negotiating trust, each with different properties with respect to speed of negotiations and caution in giving out credentials and security policies. In the autonomous world of the Internet, entities will want the freedom to choose negotiation strategies that meet their own goals, which means that two strangers who negotiate trust will often not use the same strategy. To date, only a tiny fraction of the space of possible negotiation strategies has been explored, and no two of the strategies proposed so far will interoperate. In this paper, we present a trust negotiation protocol, TrustBuilder, that acts as middleware to allow two parties to independently select their negotiation strategies. We prove that TrustBuilder allows a very large class of trust negotiation strategies to interoperate in a safe and complete manner. We also present three new families of trust negotiation strategies, demonstrate the interoperability of two of them within TrustBuilder, and show that the third family discloses as little sensitive information as possible, in a certain sense, as any strategy that can be used with TrustBuilder.

1. Introduction

With billions of users on the Internet, most interactions will occur between strangers, i.e., entities that have no pre-existing relationship and may not share a common security domain. In order for strangers to conduct secure transactions, a sufficient level of mutual trust must be established to the satisfaction of both parties. When an interaction requires that mutual trust be established, the *identity* of the participants (e.g., their social security number, fingerprint, institutional tax ID) will often be irrelevant to determining whether or not they should be trusted. Instead, the *properties* of the participants will be most relevant. For example, a service may screen potential clients based on their employment status, citizenship, age, group membership, or some other property. Traditional security approaches based on identity require a new client to pre-register with the service, in order to obtain a local login, capability, or credential before requesting service; but the exact same problem arises when the client needs to prove on-line that he, she, or it is eligible to register with the service. Further, the inconvenience of off-line and even on-line manual pre-registration can be a deterrent to clients and a costly expense for servers, who must record information about all registered clients. A less costly, more scalable approach that allows automatic on-line pre-registration, or does away entirely with the need for pre-registration, will provide a significant boon to e-commerce in the future, by allowing substantial growth in the number of sensitive business processes that can be accomplished on line. We believe that automated trust establishment is such a solution.

With automated trust establishment, strangers establish trust by exchanging *digital credentials* that provide each participant with information from what others have to say about their counterpart. Digital credentials are the on-line analogues of paper credentials that people carry in their wallets: digitally signed assertions by a credential issuer about the credential owner. A credential is signed using the issuer's private key and can be verified using the issuer's public key. A credential describes one or more attributes of the owner, using attribute name/value pairs to describe properties of the owner asserted by the issuer. Each credential also contains the public key of the credential owner. The owner can use the corresponding private key to answer challenges or otherwise demonstrate ownership of the credential. The owner can also use the private key to sign another credential, owned by a third entity.

Thus, credentials may be combined into *chains*, where the owner of one credential is the issuer of the next credential in the chain. Credential chains permit one entity to trace a web of trust from a known entity, the issuer of the first credential in the chain, to the submitting entity in which trust needs to be established. Multiple chains can be submitted to demonstrate additional properties of the submitting entity and its relationships with known entities. For example, one credential chain might be used to establish that the server is a member of the Better Business Bureau of the USA and another chain might be used to demonstrate that the server has agreed to safeguard private information.

While some resources are freely available to all, many require protection from unauthorized access. These “protected” resources should be governed by access control policies that specify the requirements to be satisfied in order to be granted access. Access control policies can be used for a wide variety of protected resources, such as services accessed through URLs, roles in role-based access control systems, and capabilities in capability-based systems. Since credentials themselves, and even access control policies, can contain sensitive information, they can also be viewed as resources whose disclosure will often also be governed by access control policies.

2. Trust negotiation

In our vision of the future of automated trust establishment, the bulk of the interactions between strangers will occur between software agents acting autonomously on behalf of human users. High-level policy directives that are established a priori by users and institutions will control the trust decisions made by software agents. These policies must adapt to a changing environment in cases of emergency or other unexpected events. These policies must be both easy to create and easy to administer. Common, proven policies will be shared among communities with similar interests and intentions. Changes to policy must proliferate rapidly through the networking infrastructure when required in order to respond to attacks or to changing security requirements.

In our approach to automated trust establishment, trust is established incrementally by exchanging credentials and requests for credentials, an iterative process known as *trust negotiation*. Different negotiation strategies determine when and how credentials are disclosed; many strategies will require that access control policies be *mobile*, that is, that the policies themselves, or some distillation of the policies, be sent to the other party in a negotiation, so that the other party can understand the requirements for gaining access to the desired resource. Credentials and policies may both contain sensitive data. Depending on the degree and nature of that sensitivity, different negotiation strategies may be appropriate in different situations.

Figure 1 presents an architecture for trust negotiation. Each participant in the negotiation has an associated security agent that manages the negotiation. The security agent mediates access to local protected resources, such as services, credentials, and access control policies. We say a credential or access control policy is *disclosed* if it has been sent to the other party in the negotiation. Disclosure of protected resources is governed by access control policies. During a negotiation, the security agent makes use of a local decision engine that accepts access control policies and credentials. Access control policies for local resources specify credentials that the other negotiation participant must provide in order to obtain access to the resource. The security agent receives credentials from the other participant and checks to see if the relevant access control policies are satisfied. The agent also requests credentials from the other party that will advance the negotiation toward the goal of granting access to the protected resource. The architecture in figure 1 supports a single protocol for establishing trust, and assumes there will be a variety of negotiation strategies that must be supported.

As noted above, when an access control policy *P* contains sensitive information, then *P* itself requires protection in the form of an access control policy for access to *P*. Such a situation requires that trust be established gradually. For example, a client interacting with an unfamiliar web server may request to see credentials that attest to the server’s handling of private information as well as certified security practices during the negotiation prior to disclosing further requests that the client considers sensitive, such as an access control policy that specifies the combination of credentials that the client requires for access to a business process that the client considers a trade secret.

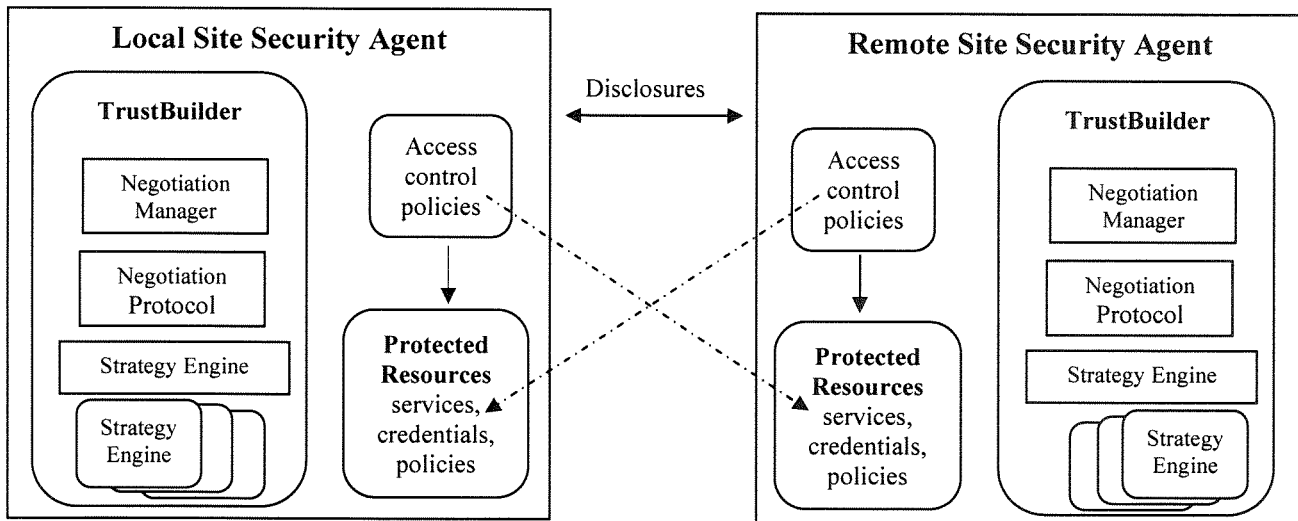


Figure 1. An architecture for automated trust negotiation. A security agent that manages local protected resources and their associated access control policies represents each negotiation participant. TrustBuilder provides the necessary middleware support to security agents to enable negotiation strategy interoperability.

2.1 Trust negotiation protocols and strategies

While a *trust negotiation protocol* defines the ordering of messages and the type of information messages will contain, a *trust negotiation strategy* controls the exact content of the messages, i.e., which credentials to disclose, when to disclose them, and when to terminate a negotiation. All trust negotiation strategies share the goal of building trust through an exchange of digital credentials that leads to obtaining access to a protected resource. Once enough trust has been established that a particular credential can be disclosed to the other party, a local negotiation strategy must determine whether the credential is relevant to the current stage of the negotiation. Different negotiation strategies will use different definitions of relevance, involving tradeoffs between computational cost, the length of the negotiation, and the number of disclosures.

Another dimension of variation is whether both parties must use the same strategy, for a successful negotiation. As discussed in the Related Work section, all of the prior work in trust negotiation assumes both parties adopt the same negotiation strategy. Furthermore, no two of the trust negotiation protocols and strategies proposed so far in the literature will interoperate. If two strangers needing to negotiate trust had each adopted a different one of these previously proposed strategies, they would be incapable of successfully negotiating trust, even if they had the requisite credentials to do so. Obviously this is impractical in the freewheeling world of the Internet, with no centralized authorities to contravene autonomy.

Lack of strategy interoperability is not the only impediment to automated trust establishment. The trust negotiation architecture shown in figure 1 includes an arrow from local access control policies to remote credentials, to indicate how local policies describe remote credentials. During a negotiation, each party may disclose policy information to the other participant that will serve as a request for credentials that are relevant to the negotiation. It is unlikely that a single language for specifying policies and requests for credentials will emerge as the lingua franca for trust negotiation. Instead, middleware that supports trust negotiation, as well as the trust strategies themselves, must be *language independent* in order to facilitate trust negotiation between a wide range of participants.

From the handful of trust negotiation strategies proposed so far in the literature, it is clear that there are endless possible variations in how to negotiate trust. Rather than exploring the space of all possible strategies one strategy (conference paper) at a time, it would be more efficient and meaningful to characterize whole groups of strategies at once and design protocols encompassing support for a broad range of potential strategies and ensuring their interoperability.

In this paper, we present *TrustBuilder*, a strategy-independent, language-independent trust negotiation protocol. We introduce three new families of trust negotiation strategies that can interoperate within TrustBuilder, and explore their properties.

3. Policies and policy graphs

When we step back to formalize the concepts of credentials and access control policies, we see a need for a more abstract representation of the information contained in credentials and policies, free of implementation details such as encryption protocols and data representation formats. In our work, we assume that the information contained in access control policies (*policies*, for short) and credentials can be expressed as finite sets of statements in a formal language with a well-defined semantics. Mathematical logic is well-suited to this purpose. For convenience, we will assume that the original language allows us to describe the meaning of a set of statements as the set of all models that satisfy the set of statements, in the usual logical sense. We say that a set X of statements *satisfies* a set of statements P if and only if P is true in all models of X .

The layers of access control policies used to guard a resource during gradual trust establishment can be represented as an *access control policy graph* (*policy graph*, for short). In this paper, a protected resource can be a service, a policy, or a credential. A policy graph for a protected resource R is a finite directed acyclic graph with a single source node S and a single sink R . (For simplicity, we assume that the name of a node is the name of the resource it represents.) All the nodes except R represent policies that specify the properties that a negotiation participant may be required to demonstrate in order to gain access to R . Every node in the graph except R is called a *policy node*. Each sensitive credential and service will have its own separate policy graph. Each policy node in a policy graph G implicitly also has its own protective graph---the maximum subgraph of G for which that policy node is the sole sink. Example policy graphs are given in the Appendix; their meaning will be discussed later.

Ordinary logic languages can be extended for use as policy languages, by extending the semantics of the language so that the meaning of “the set X of statements satisfies the policy at policy graph node N ” is defined with respect to a *path through the policy graph to node N* , rather than only the statements in the policy at N . This is because the safety of the disclosure of a resource may depend on the simultaneous satisfaction of many access control policies, and these policies may need to share references to variables. For example, one potential policy representation language is first-order logic without quantifiers or negation, with the following semantics: suppose that a party would like to know if it would be safe to disclose policy P_n , i.e., whether P_n is satisfied by the set W of credentials disclosed so far by the other party. W satisfies P_n if and only if in P_n ’s policy graph, there is a path $P_1, \dots, P_n$ from the source of the graph to P_n such that if $X_1, \dots, X_m$ are the free variables in policies $P_1, \dots, P_{n-1}$, then W satisfies the formula $\exists X_1 \dots \exists X_m (P_1 \wedge \dots \wedge P_{n-1})$ under the usual first-order semantics. While the policies in a single policy graph should be written in a single language, it is perfectly acceptable for different policy graphs to use different languages.

For the remainder of the paper, we require that the language used to represent policies and credentials be propositional, with the semantics that the policy at a policy node is satisfied by a set of formulas W if and only if there is a path from the source of the graph to that policy node, such that all the formulas in all the policy bodies along that path are true in all models of W . For purely practical reasons, we also require that the language be *monotonic*, i.e., if a set of statements X satisfies P , then any superset of X will also satisfy P ; that way, once a negotiation strategy has determined that the credentials disclosed by a participant satisfy the policy at a particular node, the strategy knows that that same policy will be satisfied for the rest of the negotiation, and does not have to be rechecked. Example suitable languages are propositional logic without negation, with the usual semantics; and propositional datalog [1], with the usual datalog semantics. These languages will already be familiar to the reader, and their semantics do not need modification (other than a restriction to monotonic subsets) for use with policy graphs. This simplicity allows us to focus on the properties of the negotiation protocol and strategy rather than on explanations of our handling of variables, negation, graph semantics, and how to determine whether a credential appears in a formula. Non-propositional monotonic languages are also readily available, e.g., in the work on stable models and the well-founded semantics from the logic programming community [1].

To prove that it satisfies the policy at node N , a negotiation participant will submit sets of credentials, whose union forms the set X that must satisfy N . For convenience, we will often say that a *negotiation participant satisfies a policy node N* (policy N , for short) if the set X of credentials provided so far by the participant

satisfies the policy at node N . A node N can have the empty set of formulas as its policy---the easiest policy to satisfy.

As mentioned earlier, if a negotiating party sends one of its credentials or policies to the other party, we say that that resource has been *disclosed*. If trust negotiation succeeds and a party is allowed to access the originally requested service R , for convenience we will also say that R has been disclosed. Algorithms for trust negotiation must ensure that every disclosure is *safe*, i.e., that it does not violate the policies put in place to protect the disclosed resource. Under our semantics for policy graphs, it is always safe to disclose the source node of a policy graph (i.e., to disclose the body of the policy at that node, along with an indication of which graph the node comes from). A party can safely disclose the body of a non-source policy node N in a policy graph if and only if there is a directed path from S to one of N 's parents in that graph, such that the other negotiation participant satisfies S with respect to that path. (We call such a path an *authorized path* to N , and say that N is *unlocked*. By convention, the source node of a policy graph is always unlocked.) R itself can be safely disclosed once there is an authorized path to R . Finally, the disclosure of an edge between two policy graph nodes is safe if the edge leads between two nodes that have been disclosed, or if the edge leads directly to the sink of the graph. If every disclosure in a negotiation is safe, we say that the negotiation itself is safe. The goal of gradual trust establishment is to find a sequence of safe disclosures that culminates in the disclosure of R . The assumption of language monotonicity means that once a node N is unlocked, the policy or credential associated with N can safely be disclosed at any point in the remainder of the negotiation.

For trust negotiation to be ubiquitous, there must be widely understood languages for representing policies and credentials. Actual credentials will be encrypted objects that follow a specified format of fields and field values. To reason about credentials at the level of a strategy engine, the actual credential must be translated into the formal language(s) used to represent the policy(s) in which that credential occurs. The translation must be the same for both negotiating parties, so that both parties can agree on whether a particular set of credentials satisfies a particular policy. A language into which credentials are translated need not resemble the credential format; for example, for a simple policy that does not need the extra expressive power of first-order logic, it would be perfectly acceptable to translate a driver's license credential from Illinois into the propositional constant IL , meaning that the negotiating party has a driver's license from Illinois. In the context of first-order logic, the same credential could be translated quite differently, e.g., $driversLicense(X)$ and $state(X) = \text{"Illinois"}$ and $name(X) = \text{"Jane Doe"}$ and $dateOfBirth(X) = \text{"1/1/2000"}$.

As just pointed out, an actual credential will be an encrypted object that looks nothing like the translated version of that credential. In our pseudocode, however, the same single propositional symbol is used both to denote the credential itself (e.g., in statements such as "send C to the other party" and "examine C 's policy graph"), and the translated version of that credential that occurs in policies (e.g., $(C \wedge C_1) \vee C_2$). We draw attention to this point because software would not confuse the two, but a human reader might. We also note that in practice, a single credential might translate into a formula involving several propositional symbols; our pseudocode would need extension to cover this possibility.

In general, negotiation strategies assume that both participants bargain in good faith. A service provider or client might wish to start its trust negotiation by verifying that its negotiation partner is using a negotiation package that has been certified by an appropriate inspection service, giving confidence that the negotiations will adhere to certain ethical guidelines.

One approach to negotiating trust between strangers that is not considered in this paper is the reliance on a trusted third party to determine whether or not two parties trust each other, according to the policies they have established. If the two parties are willing to disclose their policies and credentials to a trusted third party, that party could use the approach outlined in this paper to determine whether or not the two parties can trust one another. The advantage of this approach would be that the two parties would not disclose sensitive policy and credential information to one another. However, it would require two strangers to agree on a trusted third party.

4. TrustBuilder: A strategy-independent protocol for negotiating trust

In this section we present and analyze a strategy-independent, language-independent protocol for trust negotiation that uses the policy graphs, language, credentials, and credential translations described in the

previous section. We assume that the credentials and policies do not change during a negotiation, and that all policies are satisfiable.

In TrustBuilder, a stateful protocol for trust negotiation, the two parties take turns disclosing policy nodes, credentials, and edges to one another. In this paper, we have eliminated enough features of TrustBuilder to make our discussion fit within the page limit for the conference; we call the resulting system Oakland TrustBuilder (OTB for short). OTB differs from TrustBuilder in the following ways:

1. The TrustBuilder environment allows the two parties to agree on what constitutes failure of the negotiation: 1, 2, 3, or 4 empty disclosure messages in a row. In OTB, the negotiations fail as soon as a single empty disclosure message has been sent.
2. The TrustBuilder environment lets the two parties decide whether they will disclose all satisfied unlocked policies, which has ramifications for policy caching and reuse. In OTB, satisfied unlocked policies do not have to be disclosed.
3. The TrustBuilder environment allows the two parties to decide in advance on the acceptable set of languages used to express policies in policy graphs. This allows a party to store versions of the (intuitively) same policy in several different languages, to enable negotiations with parties that understand a broad array of languages. In OTB, we assume that a single language is used to express all policies (any monotonic subset of propositional logic), and that this language is chosen before negotiations start.
4. The TrustBuilder environment allows the two parties to decide what kind of edge disclosures to make. The options are: do/do not disclose only those edges that actually appear in a local policy graph; do/do not disclose an edge between a disclosed policy node and a locked policy graph sink node; do/do not disclose every edge (N, M) , such that N and M are local disclosed policy nodes, and there is a directed path from N to M in a local policy graph. OTB does not allow the first option, which only makes sense when all satisfied policies are disclosed; and OTB assumes that the parties have agreed on their edge disclosure options before negotiations start. Each party can use a different edge disclosure option.

The pseudocode for OTB is given in figure 2. When a client requests a service, the server first checks to see if the service is freely available, i.e., if its access control policies are always satisfied. If so, the client is immediately granted access to the server. Otherwise, the server sets the global variable *RequestedService* to

```

OTB_handle_disclosure_message(Credentials, Policies, Edges)
Input: Sets Credentials, Policies, and Edges contain the newly disclosed remote credentials, remote policy nodes, and remote
edges from the other party.
OTB_check_for_failure(Credentials, Policies, Edges). // Stop negotiating, if appropriate.
OTB_find_something_to_disclose(Credentials, Policies, Edges).
End of OTB_handle_disclosure_message.

OTB_find_something_to_disclose (Credentials, Policies, Edges)
// First, let the strategy engine decide what the new disclosures should be.
(Credentials, Policies, Edges) = Local_strategy_engine(Credentials, Policies, Edges).
Send a disclosure message containing (Credentials, Policies, Edges) to the other party.
OTB_check_for_failure(Credentials, Policies, Edges). // Stop negotiating, if appropriate.
End of OTB_find_something_to_disclose.

OTB_check_for_failure (Credentials, Policies, Edges)
If Credentials, Policies, and Edges are all empty sets,
    Then negotiations have failed. Stop negotiating and exit.
End of OTB_check_for_failure.

```

Figure 2. Pseudocode for the OTB protocol for trust negotiation.

designate the requested service. This global variable is used to determine whether trust negotiation has succeeded. The server then begins trust negotiation by calling `OTB_find_something_to_disclose( $\emptyset, \emptyset, \emptyset$ )`. `OTB_find_something_to_disclose( )` calls the local strategy engine to decide what to disclose first, typically the

source node of the policy graph for *RequestedService*. More generally, it is the strategy engine's job to determine what is unlocked (using calls to the language-dependent modules) and decide what to disclose at each round of the negotiation. *OTB_find_something_to_disclose()* sends the new disclosures to the other party and then checks to see if the negotiation has failed.

The party receiving a disclosure message handles it using *OTB_handle_disclosure_message()*, *OTB_handle_disclosure_message()* first checks to see if negotiations have failed (i.e., an empty disclosure message) and then updates the state of the negotiation by recording the newly disclosed credentials, policies, and edges. Then the routine calls *OTB_find_something_to_disclose()* to determine what to disclose next. The process repeats until negotiations fail or *RequestedService* itself is disclosed. Examples of OTB in action are given in the Appendix; the examples rely on the instantiations of the strategy engine described in the next section.

Credential and edge disclosures are straightforward. Policy node disclosures are more complex; each must include the body of the policy associated with the node, tell what credential the node is protecting, and give a unique identifier to the node, so that its associated edges can be disclosed. Our pseudocode assumes that all this information is available for each disclosed policy node, without giving any detail about the syntax used for the policy node disclosure.

OTB never overrules a disclosure recommendation made by a strategy engine; fancier protocols might second-guess a disclosure recommendation, compare suggestions from multiple strategy engines, or change strategy engines during a negotiation. Protocols that do this must record the state of the negotiations, rather than relying on the strategy engines to do so.

Not every negotiation strategy is suitable for use with every negotiation protocol; each protocol must have a set of criteria that must be satisfied by every strategy engine used with the protocol. A strategy engine is *approved for OTB* if the strategy engine satisfies the following criteria:

1. Each disclosure suggested by the strategy engine must be safe. (It follows that every OTB disclosure is safe.)
2. The strategy engine must be *complete for OTB* with the selected edge disclosure option. The completeness criteria for OTB are very complex, and we defer their presentation to Section 6.2.
3. The strategy engine must never suggest the disclosure of the same credential, policy node, or edge twice.
4. The strategy engine must immediately disclose exactly those edges that are safe to disclose and that, according to the chosen edge disclosure option, can be disclosed.

Theorem 1. *Suppose two negotiating parties use OTB with approved strategy engines S_1 and S_2 , respectively. Then the two parties will eventually disclose the requested service if and only if there is a safe sequence of credential disclosures culminating in the disclosure of the requested service. ♦*

Theorem 2. *Let c_1 be the total number of nodes in the policy graphs for the credentials possessed by the client, and let c_2 be the total number of nodes in the policy graphs for the credentials possessed by the server, plus the number of nodes in the policy graph for the originally requested service. Then a negotiation conducted with OTB will always terminate after at most $2*c_2 - 1$ messages, if $c_2 < c_1 + 1$, or after $2*c_1 + 2$ messages, if $c_2 \geq c_1 + 1$. ♦*

Because OTB never discloses the same item twice, the total size of all messages is limited to the size of the stored policies and credentials of the two parties, plus the size of the empty final disclosure message, if any.

5. Language-dependent policy analysis routines

Strategy engines need language-dependent modules that can be called to find out whether a particular policy node is unlocked by a certain set of credentials. For this purpose, we use a routine called *O_satisfied(PolicyNode, Credentials)*, where *PolicyNode* is a policy node and *Credentials* is the set of credentials which should be used to try to unlock *PolicyNode*. *O_satisfied()* (the Oakland version of Satisfied) is a strategy-independent, protocol-independent, language-dependent module that can check to see which policy language is being used for this negotiation and call appropriate routines to interpret the body of a policy,

translate credentials into that language, and test whether a policy node is unlocked. Because of our assumptions about language monotonicity and immutability of credentials and policies, the implementation of `O_satisfied()` can cache information about which policy nodes are unlocked, if desired. The nature of the satisfiability test depends on the semantics of the language, which in general must be defined over policy graphs, rather only over single policies; thus `O_satisfied()` may need to traverse part of a policy graph, so its implementation does in general depend on the data structure used to represent policy graphs. `O_satisfied()` can be used to see if remote disclosed policy nodes are unlocked by certain sets of local credentials, and to see whether local policies are unlocked by sets of remote disclosed credentials.

One other language-dependent routine, `O_appears(PolicyNodes)`, is needed for this paper. `O_appears()` parses the policy of each node in *PolicyNodes* and returns the set of all credentials that appear in those policies.

The language-dependent aspects of trust negotiation can and should be confined to two small portions of the trust negotiation system: (1) the routine that tests satisfaction and its helpers such as `O_appears()`, and (2) the routines that translate actual credentials into statements in the policy representation language.

6. Language-independent negotiation strategies

While all negotiation strategies share the goal of safely disclosing the originally requested service, they differ in how they try to construct an authorized path to the node representing the service, how they define what resources are relevant to that path, and how quickly they disclose unlocked relevant resources. Perhaps surprisingly, it is perfectly reasonable for a party to disclose edges and policies that do not actually exist, to omit disclosure of some edges and policies that do exist, within certain limits, and to disclose the policy for a credential that the party does not possess. In this paper, we do not have room to present all the edge and policy disclosure options, much less explore their surprisingly deep ramifications for OTB strategy engine completeness criteria. Instead, in this paper we require parties to disclose the exact bodies of policies, disclose no edges, and disclose policies only for credentials that they actually possess.

Negotiation strategies differ in how they manage the tradeoff between quickly obtaining access to the desired service, and minimizing the disclosure of information. In general, the harder a strategy tries to minimize disclosure, the more complex and computationally expensive it will be. Negotiation strategies that must work with tight deadlines, limited power supplies, or limited communication ability will need to avoid the cleverness used by other strategies to attempt to minimize disclosure. A strategy could measure “minimality” of disclosure by counting the number of disclosed resources, using subset inclusion, adding up sensitivity weights assigned to disclosable resources, minimizing the maximum sensitivity level of the disclosed resources, or many other possibilities.

In the limited space of this conference paper, we cannot present and analyze very many strategy engines. We will confine ourselves to two strategies and two families of strategies, none of which is equivalent to any previously published negotiation strategy. The first, called *Simple*, is fast and free with its disclosures. The second, *NotSoSimple*, is more conservative in its disclosures, and may take longer to choose its disclosures. The families of strategies are called the *Datalog Negotiation State Description* strategies and the *Derivation Tree Strategies*. Under certain assumptions, the members of the latter family are as stingy in their disclosures as it is possible for any complete-for-OTB strategy to be. In fact, we use this family to define the completeness criteria imposed by OTB on its strategy engines.

6.1 The *Simple* and *NotSoSimple* strategy engines

The *Simple* strategy engine, given in figure 3, discloses every unlocked local credential and every unlocked policy node that protects a locked local credential. At startup, *Simple* puts the source nodes of all its policy graphs into a queue, and then calls `Simple_process_queue()` to determine which policies in the queue are actually satisfied. Any satisfied policy will have all its child policy nodes unlocked and added to the queue. If a graph sink node is unlocked, the credential (or service, in the case of *RequestedService*) associated with that node is disclosed. *Simple* uses a static variable called *Boundary* to remember what unlocked nodes that protect a locked local credential may have locked children. After startup, *Simple* calls `Simple_process_queue()` with every member of *Boundary* in the queue.

The NotSoSimple strategy engine, given in figure 4, has the same overall structure as the Simple strategy engine, and uses the same queue processing code. However, NotSoSimple takes additional steps to try to make sure that its disclosures might actually move the negotiation closer to success. Rather than disclosing all unlocked policies, NotSoSimple discloses only the innermost unlocked policies of credentials that appear in a

```

Simple_strategy_engine(Credentials, Policies, Edges)
// Input: Credentials, Policies, and Edges are the newest disclosures from the other party. Credentials contains remote credentials, used to
// satisfy local policies. Policies contains remote policy graph nodes; those policies are to be satisfied using local credentials. Edges contains
// edges from remote policy graphs.
// Output: Three sets of local information: CredentialsToDisclose, PoliciesToDisclose, and EdgesToDisclose.
// Philosophy: Disclose everything that is unlocked.
// The engine maintains its state using the static variables Boundary, RemoteCredentials, RemotePolicies, UnlockedLocalNodes,
// DisclosedLocalCredentials, and DisclosedLocalPolicies (all initialized to the empty set before negotiation starts).
// Boundary is the set of all unlocked policy nodes that (as far as we know) have a locked child. RemoteCredentials and RemotePolicies contain
// the disclosures made by the other parties. UnlockedLocalNodes records which local nodes we know to be unlocked, and
// DisclosedLocalCredentials and DisclosedLocalPolicies record the local credentials and policy nodes, respectively, that have been disclosed.
RemoteCredentials = RemoteCredentials  $\cup$  Credentials. RemotePolicies = RemotePolicies  $\cup$  Policies.
Create queue Queue containing every policy node in Boundary.
NewlyUnlockedSources =  $\emptyset$ .
If negotiations have just started,
    Then add the source node of every local policy graph to Queue, UnlockedLocalNodes, Boundary, and NewlyUnlockedSources.
// Now see if any of these policies are now satisfied.
(NewlyUnlockedSinks, NewlyUnlockedPolicies) = Simple_process_queue(Queue, RemoteCredentials, Boundary, LocalUnlockedNodes).
For each credential C such that the sink node N of its policy graph is in NewlyUnlockedSinks // Disclose all unlocked local credentials.
    If C is not in DisclosedLocalCredentials, then add C to CredentialsToDisclose and DisclosedLocalCredentials.
For each member N of NewlyUnlockedPolicies  $\cup$  NewlyUnlockedSources // Disclose all unlocked policies.
    If N is not in DisclosedLocalPolicies, then add N to PoliciesToDisclose and DisclosedLocalPolicies.
EdgesToDisclose = OTB_find_edges_to_disclose(CredentialsToDisclose, PoliciesToDisclose, DisclosedLocalCredentials, DisclosedLocalPolicies).
Return (CredentialsToDisclose, PoliciesToDisclose, EdgesToDisclose).
End of Simple_strategy_engine.

Simple_process_queue(Queue, Credentials, Frontier, UnlockedLocalNodes)
// Input: A queue of policy nodes whose policies might now be satisfied by Credentials.
// Frontier is the set of all policy nodes that are unlocked and we think might now be satisfied.
// Output: The sets NewlyUnlockedSinks and NewFrontier, initially empty. Frontier and UnlockedLocalNodes will be updated.
Until Queue is empty
    Remove a policy node N from the head of Queue.
    If O_satisfied(N, Credentials) == true
        Then // N's policy is satisfied by the credentials, so N's children are now unlocked.
            Remove N from Frontier.
            If N has an outedge to the sink of this policy graph,
                Then // We have unlocked the credential at the sink of the graph, or unlocked RequestedService.
                    If the sink is the node representing RequestedService,
                        Then negotiations have succeeded. Exit and provide access to RequestedService.
                    Else // Stop looking for authorized paths in the policy graph with sink N.
                        Add the sink node to NewlyUnlockedSinks and UnlockedLocalNodes.
                        Remove from Queue and Frontier all nodes from the same graph as N.
            Else // N has children that are also policy nodes.
                For each child NC of N
                    If NC isn't already in UnlockedLocalNodes, add NC to Queue, UnlockedLocalNodes,
                    NewlyUnlockedPolicies, Frontier, and NewFrontier.
Return (NewlyUnlockedSinks, NewFrontier).
End of Simple_process_queue.

```

Figure 3. Pseudocode for the Simple strategy engine.

policy disclosed by the other party. If the other party discloses enough credentials for there to be an authorized path to the sink of a local policy graph containing policy nodes that Simple has disclosed, then Simple discloses the credential protected by that policy graph, as long as disclosure of that local credential will allow at least one additional disclosed remote policy P to be satisfied, and the remote credential C that is protected by P has still not been disclosed. On the other hand, NotSoSimple never checks whether there is still a need for the remote credential C to be disclosed; perhaps the local policy in which C appears has already been satisfied using other disclosed remote credentials.

Simple and NotSoSimple ignore all disclosures of remote edges, and they both work with any local edge disclosure option. They call the utility routine `OTB_find_edges_to_disclose()`, supplied with OTB, to construct their own edge disclosures. `OTB_find_edges_to_disclose()` finds every local edge that can safely be disclosed, has not yet been disclosed, and should be disclosed according to the chosen edge disclosure option. The pseudocode for `OTB_find_edges_to_disclose()` is very straightforward, and we do not present it here.

The Appendix contains a complete example of Simple (the client) negotiating with NotSoSimple (the server). We jump the gun slightly and present a theorem that cannot be proved until we define the completeness

```

NSS_strategy_engine(Credentials, Policies, Edges)
// Input: Credentials, Policies, and Edges are the newest disclosures from the other party. Credentials contains remote credentials, used to satisfy
// local policies. Policies contains remote policy graph nodes; those policies are to be satisfied using local credentials.
// Edges contains edges from remote policy graphs.
// Output: Three sets of local information: CredentialsToDisclose, PoliciesToDisclose, and EdgesToDisclose.
// The engine's state is in the static variables Innermost, RemoteCredentials, RemotePolicies, UnlockedLocalNodes, DisclosedLocalCredentials,
// and DisclosedLocalPolicies (all initialized to the empty set before negotiation starts). RemoteCredentials and RemotePolicies contain the
// other party's disclosures. UnlockedLocalNodes records which local nodes we know to be unlocked; DisclosedLocalCredentials and
// DisclosedLocalPolicies record the local credentials and policy nodes, respectively, that have been disclosed. Innermost is the set of all innermost
// unlocked policy nodes in relevant local policy graphs that protect a locked credential.
RemoteCredentials = RemoteCredentials  $\cup$  Credentials. RemotePolicies = RemotePolicies  $\cup$  Policies.
// Does the other party now satisfy any additional local policies?
// --Perhaps, if the policy bodies contain a newly disclosed remote credential.
Create queue Queue containing every policy node in Innermost whose body contains a credential that appears in Credentials.
// --Perhaps, if negotiations are just starting.
If the local party owns RequestedService and the source node of RequestedService's policy graph is not in UnlockedLocalNodes,
    Then add the source node of RequestedService's policy graph to Queue, UnlockedLocalNodes, Innermost, and NewlyUnlockedSources.
// --Perhaps, if the other party has just mentioned a local credential for the first time.
If there is any locally owned credential  $C$  such that  $C$  appears in Policies and the source node of  $C$ 's policy graph is not in UnlockedLocalNodes,
    Then add the source node of  $C$ 's policy graph to Queue, UnlockedLocalNodes, Innermost, and NewlyUnlockedSources.
// Now see if any of these policies are now satisfied.
(NewlyUnlockedSinks, NewInnermost) = Simple_process_queue(Queue, RemoteCredentials, Innermost, UnlockedLocalNodes).
For each policy node  $N$  in Innermost that comes from a graph with a policy node in NewInnermost or NewlyUnlockedSources,
    If every path from  $N$  to the sink of the graph passes through another member of Innermost,
        Then remove  $N$  from Innermost.
For each credential  $C$  whose policy graph sink is in NewlyUnlockedSinks
    // Disclose each unlocked local credential appearing in a disclosed remote policy, unless ...
    Add  $C$  to CredentialsToDisclose.
DRP = the set containing all members  $N$  of DisclosedRemotePolicies such that O_satisfied( $N$ , DisclosedCredentials) = false
    and O_satisfied( $N$ , DisclosedCredentials  $\cup$  CredentialsToDisclose) = true and the credential to whose
    policy graph  $N$  belongs is not in DisclosedRemoteCredentials.
For each credential  $C$  in CredentialsToDisclose // ... unless that credential doesn't help satisfy an additional policy.
    If for all members  $N$  of DRP, O_satisfied( $N$ , DisclosedLocalCredentials union CredentialsToDisclose - { $C$ }) = true,
        Then remove  $C$  from CredentialsToDisclose.
DisclosedLocalCredentials = DisclosedLocalCredentials  $\cup$  CredentialsToDisclose.
For each member  $N$  of Innermost // Disclose only the innermost unlocked policies from relevant policy graphs.
    If  $N$  is not in DisclosedLocalPolicies,
        Then add  $N$  to PoliciesToDisclose and DisclosedLocalPolicies.
EdgesToDisclose = OTB_find_edges_to_disclose(CredentialsToDisclose, PoliciesToDisclose, DisclosedLocalCredentials, DisclosedLocalPolicies).
Return (CredentialsToDisclose, PoliciesToDisclose, EdgesToDisclose).
End of NSS_strategy_engine.

```

Figure 4. Pseudocode for the NotSoSimple strategy engine.

criteria for OTB's strategy engines, in a later section.

Theorem 3. *Simple and NotSoSimple are approved for use with OTB.* ♦

6.2 The *Derivation Tree Strategies* family of strategies, and completeness criteria for OTB

This section of the paper is particularly dense with definitions, theorems, and propositions. To ease the reader's path through the section, we begin by summarizing the section. The Derivation Tree Strategies (DTS) family of strategies is inspired by the observation that policy graphs can be rewritten as datalog [1] formulas containing special symbols to represent remote policies that have not yet been disclosed. Local and remote credentials can also be represented as datalog statements. At a particular point in the negotiations, we will know that the negotiation will definitely eventually succeed if we can derive R (the propositional symbol representing the originally requested service) from a certain subset of the current set of datalog statements. Disclosures of credentials and policies that do not occur in any derivation tree for R are completely unnecessary, as they can never contribute to a successful negotiation; this means that negotiations must eventually fail if there is no derivation tree for R . Further, many derivation trees are redundant and can be ignored entirely. The remaining derivation trees can also give us minimum bounds on what must be disclosed next. These observations lead to a definition of DTS and allow us to prove that under some mild restrictions, DTS is as stingy with its disclosures as it is possible to be. We also use the ideas behind DTS to define what it means for a strategy engine to be complete for OTB.

The datalog version of policies and credentials reflects only *a single party's view* of the negotiation, not an omniscient summary. Thus before we turn policies into datalog statements, we need to ensure that the remote policy nodes and edges that have been disclosed so far form a connected policy graph. Up to this point in the paper, we have been able to present results that do not depend on the option chosen for edge disclosure. Unfortunately, the edge disclosure choice determines how we add enough edges and nodes to the disclosed remote policy nodes and edges to form a connected graph, and we do not have enough space to explain how to add edges and nodes for all the disclosure options. Thus we will only show how to add edges and nodes for the case *where no remote edges are disclosed*. The results in this section also hold as stated for the other edge disclosure options.

To turn the disclosed remote policy nodes into a properly formed policy graph when no remote edges are being disclosed, first we add a remote policy graph sink node *Sink* for each remote credential appearing in a local policy, and for the originally requested service, if it is not a local service. To match each *Sink* node, introduce an additional remote node *Source*, whose policy is the empty set of formulas. We add edges from *Source* to every disclosed remote policy node that belongs to that graph. Then for each disclosed remote policy node N , we introduce a single new *hypothetical* remote policy node M whose body is the empty set of formulas, and add edges (N, M) and $(M, Sink)$ to the graph. If there are no disclosed remote policy nodes for this graph, then we create a hypothetical remote policy node O with an empty set of formulas as its body, and add edges $(Source, O)$ and $(O, Sink)$. The result is a properly formed policy graph---although quite different from the version owned by the other party. Intuitively, the hypothetical policy nodes represent the fact that we really don't know what undisclosed policies the other party has. We place a hypothetical policy node at each point in the policy graph where our access to *Sink* might be prevented by an unsatisfied, undisclosed remote policy. We give each hypothetical node an empty policy body because that is the most optimistic assumption we can make---that we have already satisfied the undisclosed policy---and we want to consider every possible way to unlock *Sink*. The second section of the Appendix gives an example of hypothetical nodes and every other construct in this section.

Then, given the set of local and (locally-constructed) remote policy graphs, disclosed remote credentials, and local credentials, we build the set of derivation trees by first rewriting that information as datalog statements:

1. Each local credential, remote credential, non-sink local policy graph node, and non-sink remote policy node is represented by a unique propositional symbol. For convenience, we will use the node or credential name as its symbol. The sink node of each policy graph must be represented by the same symbol as the credential or service protected by that graph. For convenience, we assume that the

originally requested service is named R . Source nodes for remote policy graphs will be named $Source_1$, $Source_2$, etc., and hypothetical nodes will be named H_1 , H_2 , etc.

2. For each policy node N and each potential minimal satisfaction set¹ $\{C_1, \dots, C_n\}$ of credentials for N , include the statement " $N \leftarrow C_1 \wedge \dots \wedge C_n$ " if N is the source node of the policy graph. If N has a parent node M in the policy graph, include the statement " $N \leftarrow M \wedge C_1 \wedge \dots \wedge C_n$ ", for each parent M of N .
3. For each sink node C , and each parent N of C , include the statement " $C \leftarrow N$ ".

We call the resulting set of statements the Datalog Negotiation State Description (DNSD for short). The Appendix gives an example DNSD and a set of *derivation trees* for a negotiation in progress. Derivation trees show every way to derive R from the DNSD. Each derivation tree has R as the root node. R 's children are the symbols in the DNSD rule used to derive R in that particular derivation. The children of R 's child N are the symbols in the body of the rule used to derive N in that particular derivation, and so on. The leaves of the tree are all rule heads with empty rule bodies (i.e., the policies that are always satisfied). If a tree does (resp. does not) contain hypothetical policy nodes, we say that it is a *hypothetical* (resp. *non-hypothetical*) tree.

Note that we are not requiring that any strategy engine actually compute the set of derivation trees; rather, it is an important mental exercise that will give us insight into the properties of all strategy engines that can work successfully with OTB. Also note that the DNSD is specific to a particular party at a particular point in the negotiation; the other party will have its own, often quite different, DNSD. Finally, note that as implied by the S in its name, the DNSD for remote policy graphs can change with each new disclosure message from the other party.

To reflect the disclosures that have already been made, we can *semi-prune* a derivation tree by pruning out all subtrees rooted at a disclosed credential.

Theorem 4. *If R has a non-hypothetical semi-pruned derivation tree, then there is a safe sequence of credential disclosures culminating in the disclosure of R . If R has no derivation trees, there is no safe sequence of disclosures terminated by the disclosure of R . ♦*

Derivation trees tell us how we might succeed in disclosing R , but they do not directly tell us what to disclose next. For this purpose, we define the set of *pruned* derivation trees: take each semi-pruned derivation tree, and prune out every non-hypothetical policy node that is satisfied by the set of disclosed credentials. Also prune out every hypothetical node that becomes a leaf in the pruned tree. Every leaf of a pruned derivation tree is an undisclosed credential. Example pruned derivation trees are given in the Appendix.

We say that a pruned derivation tree has a *qualifying leaf* if it has a leaf that belongs to the other party and that the other party might reasonably be expected to disclose. More precisely, a pruned derivation tree has a qualifying leaf if and only if R is a leaf of the tree, or the tree has a leaf that is a remote undisclosed credential and the nearest ancestor of that leaf that is a policy node (if such an ancestor exists) is a local disclosed policy node. If a pruned derivation tree has a qualifying leaf, we say that the tree is a *qualifying tree*. (Note that as always, all the disclosures must be safe; it is cheating to create a qualifying tree by making an unsafe disclosure.) Intuitively, the pruning process tells us which policy nodes and credentials to disclose next. Suppose we prune the derivation trees before deciding what to disclose. Then in each pruned tree that does not already qualify, we must disclose an additional leaf (our local unlocked credentials) or local policy node (closest ancestor of remote credentials in the tree) to make the tree qualify. Intuitively, a tree qualifies if the other party can safely disclose something from the tree, *and* can tell that it would be to its advantage to make that disclosure.

If T_1 is a qualifying pruned tree, we say that T_1 *covers* pruned derivation tree T_2 if, roughly speaking, negotiation progress in T_1 will inevitably lead to negotiation progress in T_2 . Covering trees are important for the case where a policy in tree T_2 entails a policy in T_1 . Often, it is not necessary to make T_2 be a qualifying tree, e.g., by disclosing the entailing policy in T_2 , because when the other party makes its next round of disclosures to qualify their version(s) of T_1 , those disclosures will automatically lead to progress with respect to the entailing policy in T_2 . For example, at the beginning of negotiations, perhaps the policy graph for R is a diamond whose

¹ If a set $X = \{C_1, \dots, C_n\}$ of credentials unlocks a policy node N , and no proper subset of X has this property, then we say that X is a minimal satisfaction set for N .

source node's policy is the empty set of formulas, and the two side points of the diamond are policies $P_1: C_1 \vee C_2$ and $P_2: C_1$. The server will have two derivation trees containing P_1 and one derivation tree containing P_2 . It suffices for the server to disclose P_1 , thereby qualifying the two trees containing P_1 , because if the other party creates a qualifying leaf in their own derivation trees that contain P_1 , the other party will either have disclosed a policy for C_1 , or will not possess C_1 . If the other party does not possess C_1 , then all derivation trees containing P_2 are irrelevant; and if the other party discloses a policy for C_1 , then the other party has cleared the way for future negotiation progress in T_2 by the local party. (If P_1 were $C_1 \wedge C_2$, then the local party could disclose just P_2 ; but if we have $P_1: C_1 \wedge C_2$ and $P_2: C_1 \vee C_3$, then the local party would need to disclose both P_1 and P_2 , even though in this case one of the three trees is covered by another tree.)

More precisely, T_1 covers T_2 if every leaf of T_1 is also a leaf of T_2 , and either (a) every hypothetical node of T_1 is also a node of T_2 ; or (b) every remote credential that appears in *any* disclosed local policy either (i) has been disclosed, or (ii) occurs in at least one place in T_2 where it has no hypothetical ancestor. Requirements (a) and (b) are present to cover the possibility that the other party does not consider T_1 to be a viable tree, because of information hidden behind nodes that are hypothetical in T_1 but are not hypothetical in the remote party's version of T_1 .

All those definitions have led to something exciting: we can finally give the completeness criterion for OTB's strategy engines! Suppose that a negotiating party has just sent a disclosure message. We say that the party's disclosure was *complete* if it satisfies the following: if there are any non-hypothetical semi-pruned derivation trees for this party immediately after the disclosure, then we require that the party have at least one non-hypothetical semi-pruned derivation tree that, after pruning, is a qualifying tree. Further, if the party has no non-hypothetical semi-pruned derivation trees immediately after the disclosure, then we require that every pruned derivation tree for this party have a qualifying leaf or be covered by a qualifying pruned tree immediately after the disclosure. Finally, we say that a strategy engine is *complete for OTB* if every disclosure that it makes is complete.

This is a good moment to recall Theorem 1, which said that negotiations using OTB with approved strategy engines would always succeed, if success was theoretically possible. We are leading up to another important theorem, showing that (almost) every strategy engine that isn't complete for OTB can cause negotiations to fail when success is possible.

At a particular point in an ongoing negotiation, suppose that a party is about to send out a disclosure message. Consider the set D of all possible safe and complete disclosure messages that the party could send out at this point. Then remove from D each disclosure message whose disclosures are a proper superset of the contents of another message in D . The remaining members of D are the *minimal complete disclosures* that the party could make.

We say that a negotiation strategy engine *belongs to the DTS family of strategies* if its disclosures are always minimal complete disclosures. We claim that the strategies in the DTS family are almost as stingy with their disclosures as it is possible to be. In the remainder of this section, we formalize and prove that claim.

A *status list* for a party, at a moment when the party has just received a disclosure message, lists the current status of each node and credential: whether it is locked or unlocked, disclosed or undisclosed. A DNSD and status list is *valid* for a particular strategy if that DNSD and strategy list could arise during an actual negotiation using that engine. We say that a strategy engine E belongs to the *DNSD family of strategies* under a particular edge disclosure option if E can be represented as a function that maps from a valid DNSD and status list to a recommended set of credential and policy node disclosures. Intuitively, E belongs to the DNSD family if E considers only the information in the DNSD and the status list when deciding what to disclose next. For example, E cannot consider the DNSD of previous states in the negotiation, or reason about the transition between two adjacent states. If E did so, E might be able to eliminate some derivation trees in the current DNSD, by figuring out that the other party does not still consider them to be viable trees. From a formal viewpoint, it is perfectly reasonable to represent and reason about the other party's beliefs during negotiation, but it requires a greater effort than we think any practical strategy engine is likely to want to expend.

Proposition 1. *DTS, Simple, and NotSoSimple belong to the DNSD family of strategies. ♦*

Confining our attention to members of the DNSD family, we say that a family F of strategies covers a strategy A if for all possible valid choices d of a DNSD and status list, there is a strategy B in F such that (1) d is valid for B , (2) the credential disclosures recommended by B for d are a subset of those recommended by A , and (3) the policy disclosures recommended by B for d are a subset of those recommended by A .

Proposition 2. *A strategy is approved for use with OTB if and only if it is covered by the DTS family of strategies.* ♦

We are now ready to show that the completeness criterion for OTB's strategy engines is necessary and sufficient, within the DNSD strategy family, to ensure that negotiations succeed whenever possible.

Theorem 5. Let S be a strategy in the DNSD family that is not approved for OTB solely because S does not satisfy OTB's completeness requirement. Then there exists a strategy E that is approved for use with OTB, a set of local and remote policy graphs and credentials, and a service request such that a negotiation using E and S will fail, even though there is a safe sequence of credential disclosures terminated by the disclosure of the requested service. ♦

7. Related work

Credential-based authentication and authorization systems fall into three groups: identity-based, property-based, and capability-based. Originally, public key certificates, such as X.509 [20] and PGP [22], simply bound keys to names, and X.509 v. 3 certificates later extended this binding to general properties (attributes). Such certificates form the foundation of identity-based systems, which authenticate an entity's identity or name and use it as the basis for authorization. Identity is not a useful basis for our aim of establishing trust among strangers. Bina et al. [2] introduced our property-based credentials to allow the binding of arbitrary attributes and support trust negotiation between strangers. Systems have emerged that use property-based credentials to manage trust in decentralized, distributed systems [9] [11] [16] [19].

Johnston et al. [11] use attribute certificates (property-based credentials) and use-condition certificates (policy assertions) for access control. Use-condition certificates enable multiple, distributed stakeholders to share control over access to resources. In their architecture, the policy evaluation engine retrieves the certificates associated with a user in order to determine if all the use conditions are met. Their work could be extended using our approach to protect sensitive certificates.

The Trust Establishment Project at the IBM Haifa Research Laboratory [9] has developed a system for establishing trust between strangers according to policies that specify constraints on the contents of public-key certificates. Servers can use a collector to gather supporting credentials from issuer sites. Each credential contains a reference to the site associated with the issuer. That site serves as the starting point for a collector-controlled search for relevant supporting credentials. Security agents in our work could adopt the collector feature, and we could use their policy definition language. Their work could be extended using our approach to protect sensitive credentials and gradually establish trust.

Capability-based systems manage delegation of authority for a particular application. Capability-based systems are not designed for establishing trust between strangers, since clients are assumed to possess credentials that represent authorization of specific actions with the application server. In the capability-based KeyNote system of Blaze et al. [3][4], a credential describes the conditions under which one principal authorizes actions requested by other principals. KeyNote policies delegate authority on behalf of the associated application to otherwise untrusted parties. KeyNote credentials express delegation in terms of actions that are relevant to a given application. KeyNote policies do not interpret the meaning of credentials for the application. This is unlike policies designed for use with property-based credentials, which typically derive roles from credential attributes. The IETF Simple Public Key Infrastructure [15] uses a similar approach to that of KeyNote by embedding authorizations directly in certificates.

The P3P standard [12] focuses on negotiating the disclosure of a user's sensitive private information based on the privacy practices of the server. Trust negotiation is generalized to base disclosure on any server property of interest to the client that can be represented in a credential. The work on trust negotiation focuses on certified

properties of the credential holder while P3P is based on data submitted by the client that are claims the client makes about itself. Support for both kinds of information in trust negotiation is warranted.

SSL [8], the predominant credential-exchange mechanism in use on the web, and its successor TLS [6][7] support credential exchange during client and server authentication. The protocol is suited for identity-based credentials and would need extension to make it adaptable to property-based credentials. Needed additions include protection for sensitive server credentials and a way for the client to explain its policies to the server.

Islam et al. [10] show how to control downloaded executable content using policy graphs. Their definition of policy graphs is different from ours, and their information that is akin to policies is not mobile, thus has no access control mechanism. Their system assumes that all the appropriate credentials accompany downloaded content. Their work could be extended using our approach to mobile policies and negotiation.

The first trust negotiation strategies proposed included a naïve strategy that disclosed credentials as soon as they were unlocked and disclosed no policy information, as well as a strategy that was its polar opposite, disclosing credentials only after each party determined that trust could be established after reviewing the other participant's policies [16]. Reference [21] introduced a new strategy that was complete and could be more efficient than the previous proposals, in a certain sense. In [14], consideration was given for sensitive policy information in several strategies that established trust gradually through the introduction of policy graphs. The fact that none of the strategies proposed in this earlier work will interoperate demonstrates the need for middleware to support interoperability between the negotiation strategies.

8. Conclusions and future work

This paper has presented a subset of TrustBuilder, a negotiation-strategy-independent, policy-language-independent piece of middleware for trust negotiation. TrustBuilder provides a large degree of autonomy to each party in choosing the negotiation strategy that best meets their needs, while guaranteeing the safety and completeness of ensuing trust negotiations. Because no two previously proposed trust negotiation strategies can interoperate, the existence of TrustBuilder increases the likelihood that the two entities can in fact negotiate trust successfully while adhering to the security requirements and negotiation style preferences that each party has already established. We believe that the introduction of this kind of middleware protocol is the single most significant enabler of ubiquitous trust negotiation.

TrustBuilder itself is an extremely simple wrapper for a strategy engine: it simply passes credential, policy, and policy graph edge disclosures back and forth between the negotiation participants. This simplicity might lead one to believe that TrustBuilder is the only possible trust negotiation protocol. Far from it! For example, previously proposed negotiation strategies that never disclose policy information are not approved for use with TrustBuilder, because the other negotiating party may be using a strategy that requires policy information to guide its choices of disclosures. In the future we will explore other possible prototypes for middleware.

TrustBuilder's simplicity is also misleading in the sense that the definition of exactly which trust negotiation strategies will work correctly with TrustBuilder was extremely challenging to derive and prove, even with our restriction to the well-behaved DNSD strategy family. It is clear that TrustBuilder will work correctly with some strategy engines that lie outside the DNSD family, and we plan to investigate these extensions in the future.

TrustBuilder grew out of our experience with the implementation of a previous framework for trust negotiation (not described in this paper due to the blind review requirements). The previous framework only worked when the two parties used the same strategy engine, and required retooling whenever we wanted to experiment with a different strategy. We will replace the old framework implementation with a new one that supports TrustBuilder.

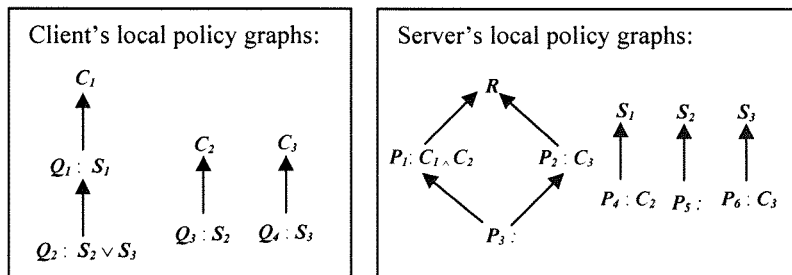
9. References

- [1] K. R. Apt, D. S. Warren, and M. Truszczyński (editors), *The Logic Programming Paradigm: A 25-Year Perspective*, Springer-Verlag, 1999.
- [2] E. Bina, V. Jones, R. McCool and M. Winslett, "Secure Access to Data Over the Internet," *ACM/IEEE International Conference on Parallel and Distributed Information Systems*, Austin, Texas, September 1994.

- [3] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust Management System Version 2," Internet Draft RFC 2704, September 1999.
- [4] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "KeyNote: Trust Management for Public-Key Infrastructures," Security Protocols, 6th International Workshop, Cambridge, UK, 1998.
- [5] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," *IEEE Symposium on Security and Privacy*, Oakland, May 1996.
- [6] T. Dierks, C. Allen, "The TLS Protocol Version 1.0," draft-ietf-tls-protocol-06.txt, Nov. 12, 1998.
- [7] S. Farrell, "TLS Extensions for Attribute Certificate Based Authorization," draft-ietf-tls-attr-cert-01.txt, Aug. 20, 1998.
- [8] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol," Netscape Communications Corp., Nov. 18, 1996.
- [9] A. Herzberg, J. 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, May 2000.
- [10] N. Islam, R. Anand, T. Jaeger, and J. R. Rao, "A flexible security system for using Internet content." *IEEE Software*, Vol. 14, No. 5, September - October 1997.
- [11] W. Johnston, S. Mudumbai, and M. Thompson, "Authorization and Attribute Certificates for Widely Distributed Access Control," *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1998.
- [12] "Platform for Privacy Preferences (P3P) Specification," W3C, <http://www.w3.org/TR/WD-P3P/Overview.html>.
- [13] B. Schneier, *Applied Cryptography*, John Wiley and Sons, Inc., second edition, 1996.
- [14] K. Seamons, M. Winslett, and T. Yu, "Limiting the Disclosure of Access Control Policies during Automated Trust Negotiation," to appear in *Network and Distributed System Security Symposium*, San Diego, CA, 2001.
- [15] Simple Public Key Infrastructure (SPKI), <http://www.ietf.org/html.charters/spki-charter.html>.
- [16] W. Winsborough, K. Seamons, and V. Jones, "Negotiating Disclosure of Sensitive Credentials", Second Conference on Security in Communication Networks, Amalfi, Italy, September 1999.
- [17] W. Winsborough, K. Seamons, and V. Jones, "Automated Trust Negotiation," *DARPA Information Survivability Conference and Exposition*, Hilton Head, January 2000.
- [18] W. Winsborough, K. Seamons, and V. Jones, "Automated Trust Negotiation," submitted for journal publication, April 2000, currently available at <http://www.csc.ncsu.edu/faculty/vej/atn.ps>.
- [19] M. Winslett, N. Ching, V. Jones, and I. Slepchin, "Using Digital Credentials on the World-Wide Web," *Journal of Computer Security*, 5, 1997, 255-267.
- [20] International Telecommunication Union, Rec. X.509 - Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, August 1997.
- [21] T. Yu, X. Ma, and M. Winslett, "PRUNES: An Efficient and Complete Strategy for Automated Trust Negotiation over the Internet," ACM Conference on Computer and Communications Security, Athens, Greece, November 2000.
- [22] P. Zimmerman, *PGP User's Guide*, MIT Press, 1994.

10. Appendix

The first section of the appendix contains a complete example of Simple (running at the client) interacting with NotSoSimple (running at the server). The second section contains a complete example of two members of the DTS family of strategies, interacting during a negotiation. The two sections use the same policy graphs, sets of credentials, and service request.



10.1 Example of *Simple* interacting with *NotSoSimple*

The following is an example of a hypothetical trust negotiation between a client and server, with the client using the Simple negotiation strategy and the server using the NSS negotiation strategy. The negotiation is based on the client and server policy graphs shown above. During each stage of the negotiation, the changes to the local state of the negotiation participant are enumerated according to the algorithms in the pseudocode presented earlier in the paper.

The client requests service *R*:

Stage 1: Server

```

Queue=<P3>;      Innermost = { P3};      UnlockedLocalNodes={ P3}
Call Simple_process_queue (<P3>, ∅, { P3}, { P3})
    Queue=<P2, P1>
    UnlockedLocalNodes = { P3, P2, R};      NewlyUnlockedPolicies = { P3, P2, P1}
    Innermost { P2, P1};      NewInnermost={ P1, P2}
    Dequeue P2 Queue=< P1>;      Dequeue P1 Queue=<∅>
    Return (∅, { P1, P2})
PoliciesToDisclose = DisclosedLocalPolicies = { P1, P2};
DisclosedLocalCreds=∅
Discloses (∅, { P1, P2}, edges)

```

Stage 2: Client

```

RemotePolicies = { P1, P2};
Queue = UnlockedLocalNodes = NewlyUnlockedSources = Boundary = { Q2, Q3, Q4}
Call Simple_Process_Queue (<Q2, Q3, Q4>, ∅, { Q2, Q3, Q4}, { Q2, Q3, Q4})
    Dequeue Q2. Queue=< Q3, Q4>;      Dequeue Q3. Queue=< Q4>;      Dequeue Q4. Queue=<∅>
    Return (∅, ∅)
NewlyUnlockedSinks = NewlyUnlockedpolicies = ∅      DisclosedLocalPolicies = { Q2, Q3, Q4}
Return (∅, { Q2, Q3, Q4}, edges)

```

Stage 3: Server

```

Credentials=∅;      RemotePolicies={ Q2, Q3, Q4};
Innermost={ P1, P2};      UnlockedLocalNodes={ P1, P2, P3}
Queue = <P5, P6>;      UnlockedLocalNodes = { P1, P2, P3, P4, P5, P6};
Innermost={ P1, P2, P5, P6};      NewlyUnlockedSources={ P5, P6}
Call simple_process_queue (<P5, P6>, ∅, { P1, P2, P5, P6}, { P1, P2, P3, P5, P6})
    Dequeue P5;      Queue=< P6>
    Frontier = Innermost = { P1, P2, P6};      NewlyUnlockedSinks={ S2}
    UnlockedLocalNodes = { P1, P2, P3, P5, P6, S2}
    Dequeue P6;      Queue=<∅>
    Return ({ S2}, ∅)
CredentialsToDisclose = DisclosedLocalCredentials = { S2}
PoliciesToDisclose={ P5, P6};      DisclosedLocalPolicies = { P1, P2, P3, P6}
Discloses ({ S2}, { P5, P6}, edges)

```

Stage 4: Client

```

RemoteCredentials = { S2};
RemotePolicies = { P5, P6, P1, P2}      Queue = < Q2, Q3, Q4>
Call Simple_Process_Queue (<Q2, Q3, Q4>, { S2}, { Q2, Q3, Q4}, { Q2, Q3, Q4})
    Dequeue Q2. Queue=< Q3, Q4>;      BoundaryFrontier={ Q2, Q3}
    Queue=< Q3, Q4, Q1>      UnlockedLocalNodes={ Q2, Q3, Q4, Q1}
    NewlyUnlockedPolicies={ Q1}      Boundary=Frontier={ Q3, Q4, Q1}
    NewFrontier={ Q1}
    Dequeue Q3. Queue=< Q4, Q1>;      Boundary = Frontier = { Q4, Q1}
    NewlyUnlockedSinks={ C2};      UnlockedLocalNodes={ C2, Q1, Q4}
    Dequeue Q4. Queue=< Q1>      Dequeue Q1. Queue=<∅>
    Return ({ C2}, { Q1})
CredentialsToDisclose = { C2} = DisclosedLocalCredentials = { C2}; DisclosedLocalPolicies = { Q1, Q2, Q3, Q4}
Disclose ({ C2}, { Q1}, edges)

```

Stage 5: Server

```

RemoteCredentials={ C2};
Queue = < P1>;
UnlockedLocalNodes = { P1, P2, P3, P4, P5, P6, S2};
Innermost={ P1, P2, P6, P4};
Call simple_process_queue (< P1, P4>, { P1, P2, P4, P6}, { P1, P2, P3, P4, P5, P6, S2})
    Dequeue P1;
    Dequeue P4;
    Frontier = Innermost = { P1, P2, P6};
    UnlockedLocalNodes = { P1, P2, P3, P4, P5, P6, S1, S2};
    Return (NewlyUnlocked{ S2}, ∅)
DisclosedLocalCredentials = { S1, S2}
Discloses ({ S1}, ∅, edges)

```

Stage 6: Client

```

RemoteCredentials = { S2, S1};
RemotePolicies = { P1, P2, P5, P6}
Queue = < Q4, Q1>
Call Simple_Process_Queue (< Q4, Q1>, { S1, S2}, { Q4, Q1}, { Q1, Q2, Q3, Q4, C2})
    Dequeue Q4. Queue=< Q1>;
    NewlyUnlockedSinks={ C1};
    Frontier = Boundary = { Q4}
    Return (NewlyUnlockedSinks: { C1}, NewlyUnlockedPolicies: ∅)
DisclosedLocalCredentials = { C1, C2};
Disclose ( { C1}, ∅, edges)

```

Stage 7: Server

```

DisclosedRemoteCredentials = { C1, C2};
Innermost={ P1};
Call simple_process_queue (< P1>, { C1, C2}, { P1}, { P1, P2, P3, P4, P5, P6, S1, S2})
    Dequeue P1;
    Frontier = Innermost = ∅
Exit Negotiation.

```

The server grants the client access to service R.

10.2 Example of the DTS family of strategies in action

The section provides an example of a hypothetical trust negotiation between a client and server using the DTS strategy. The negotiation is based on the client and server policy graphs presented at the beginning of this section. The following boxes contain the DNSD graphs for the client and server.

Client's DNSD for local policy graphs

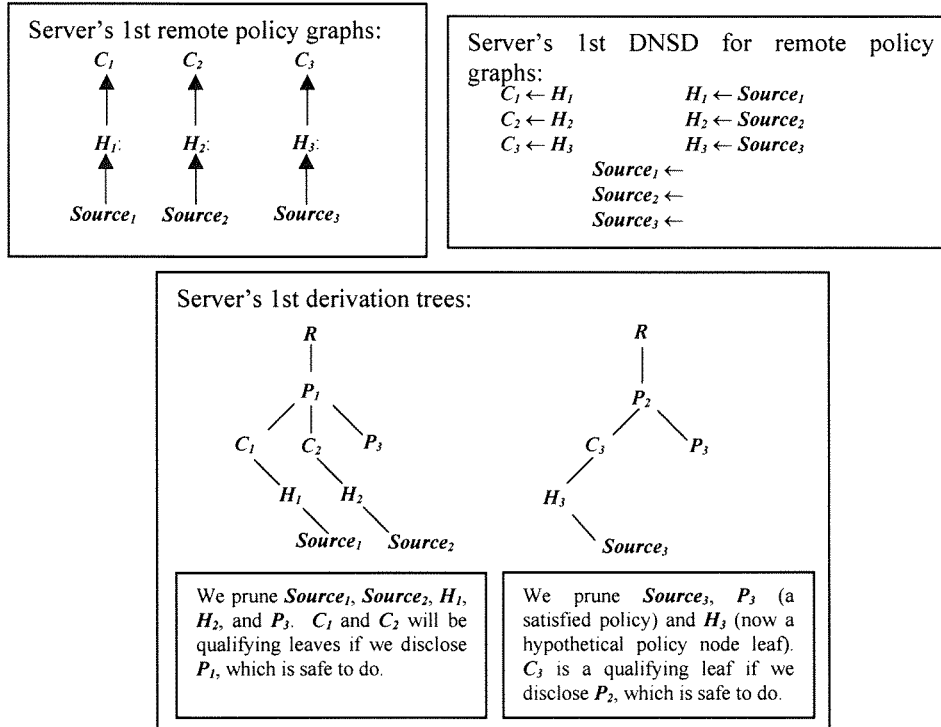
$C_1 \leftarrow Q_1$	$Q_1 \leftarrow Q_2 \wedge S_1$
$C_2 \leftarrow Q_3$	$Q_2 \leftarrow S_3$
$C_3 \leftarrow Q_4$	$Q_4 \leftarrow S_3$

Server's DNSD for local policy graphs

$R \leftarrow P_1$	$P_1 \leftarrow P_3 \wedge C_1 \wedge C_2$
$R \leftarrow P_2$	$P_2 \leftarrow P_3 \wedge C_3$
$S_1 \leftarrow P_4$	$P_3 \leftarrow$
$S_2 \leftarrow P_5$	$P_4 \leftarrow C_2$
$S_3 \leftarrow P_6$	$P_5 \leftarrow$
	$P_6 \leftarrow C_3$

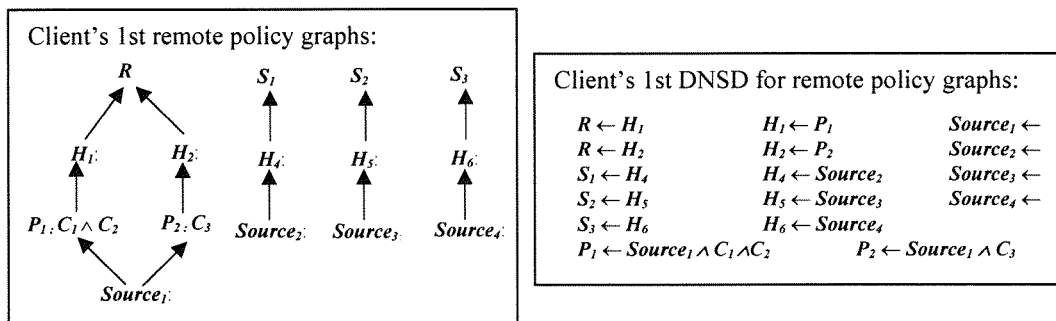
The Client requests service R .

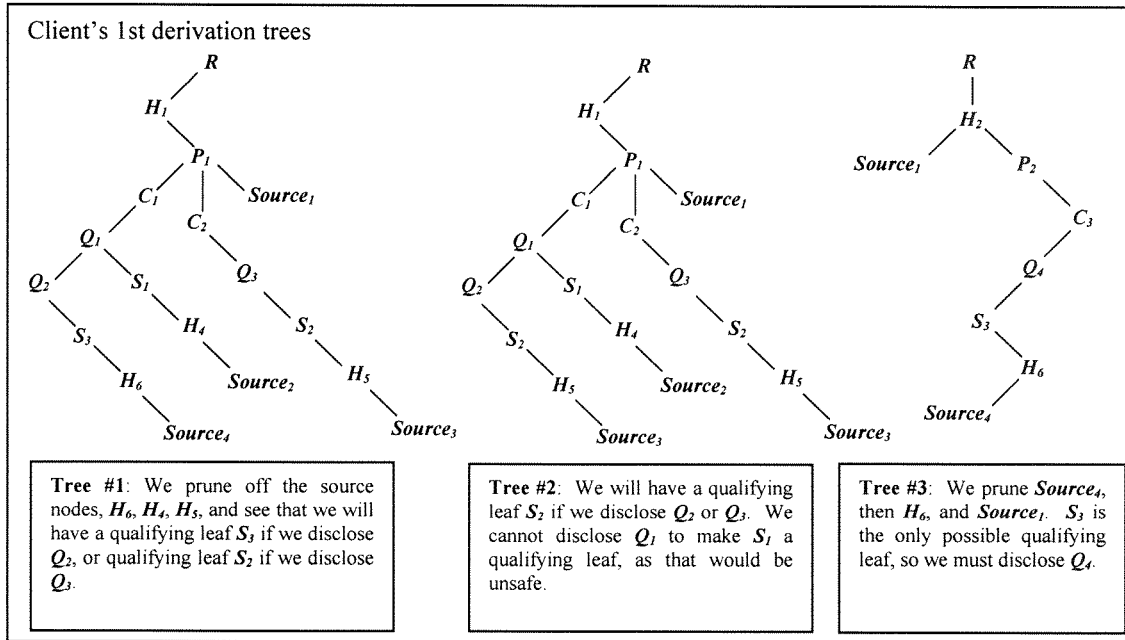
Stage 1: Server



The server discloses P_1 and P_2 .

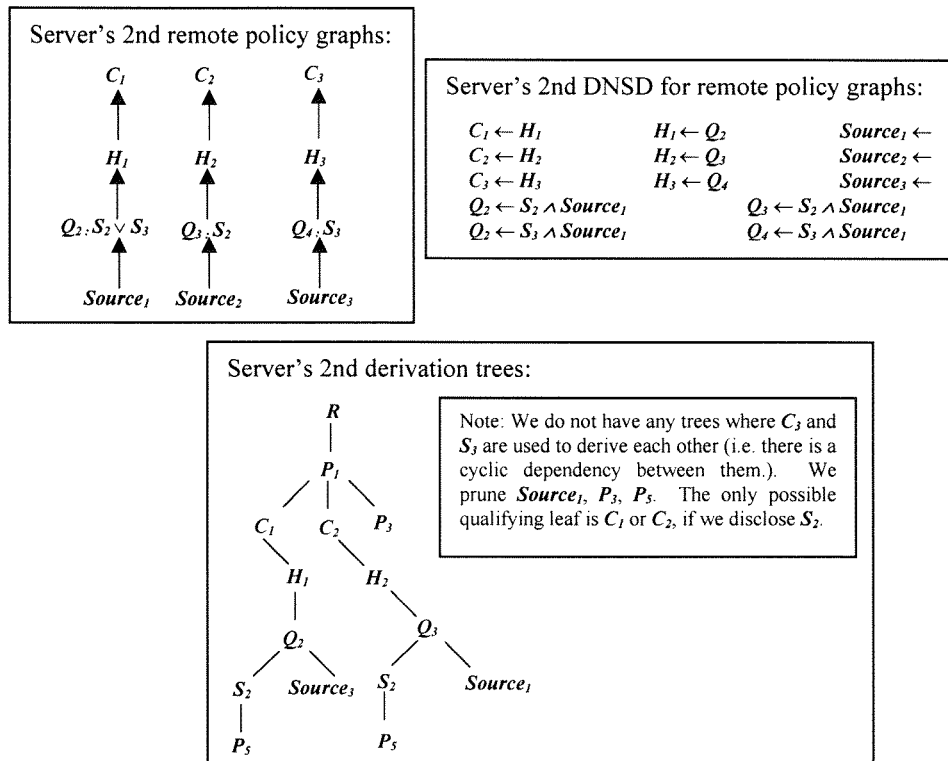
Stage 2: Client





The client chooses to disclose Q_4 and both Q_2 and Q_3 (either one alone of Q_2 and Q_3 would suffice, but this should move negotiations along faster).

Stage 3: Server



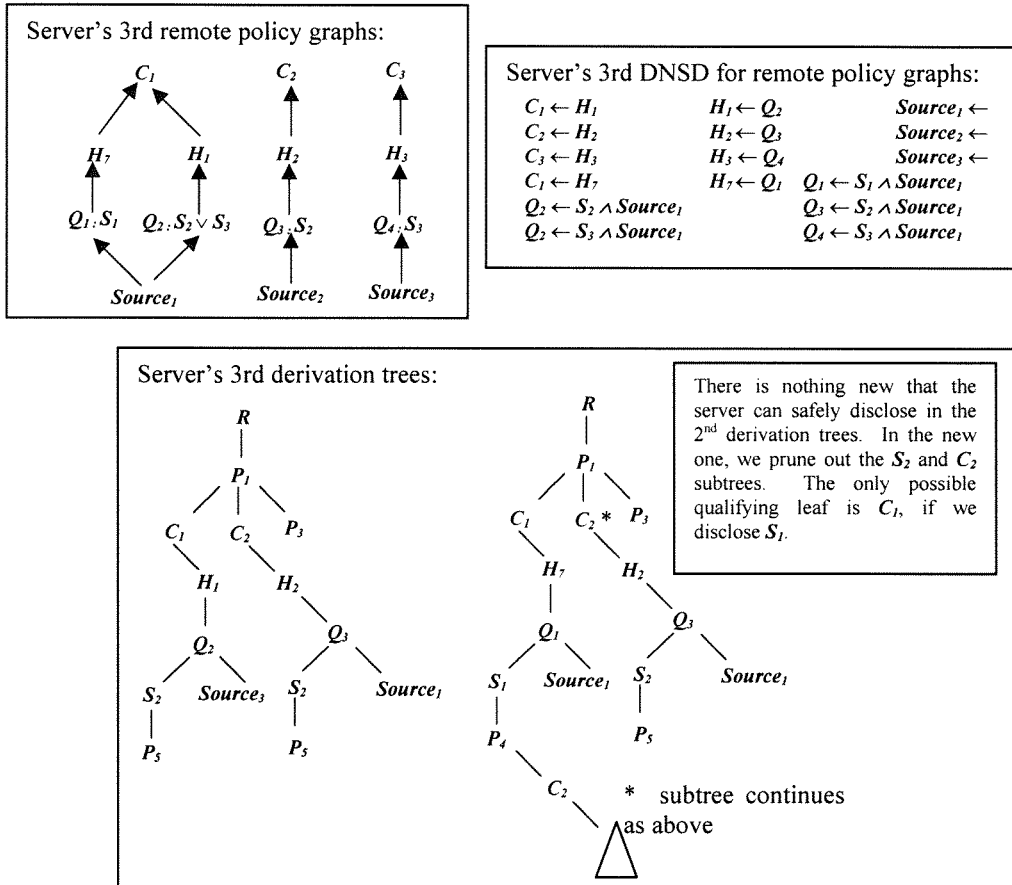
The server discloses S_2 .

Stage 4: Client

The client's 2nd remote policy graphs, 2nd DNSD, and 2nd derivation trees are all unchanged from the 1st versions in stage 2. We can now prune out all subtrees rooted at S_2 . For tree #1, S_1 is already a qualifying leaf. We could also disclose C_2 if we want to speed up negotiations. S_3 is also a qualifying leaf. For tree #2, the only potential qualifying leaf is S_1 , if we disclose Q_1 . If desired, we can also disclose C_2 , if we want to speed up negotiations. For tree #3, S_3 is already a qualifying leaf, and there is nothing else we can disclose. Conjecture: we can delete #1 and #3 due to lack of change.

The client chooses to disclose Q_1 and C_2 .

Stage 5: Server



The server discloses S_1 .

Stage 6: Client

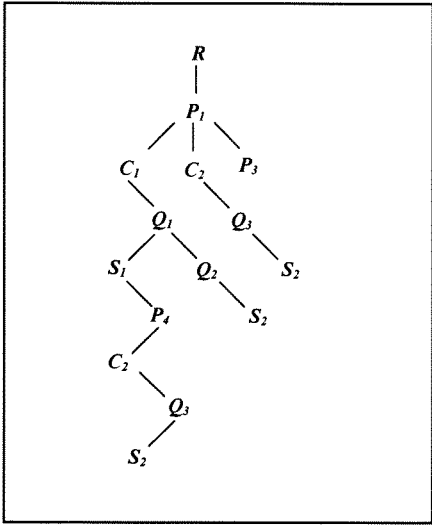
The client's 3rd remote policy graphs, DNSD, and derivations are unchanged (see 1st ones above), but we can prune higher, removing subtrees rooted at S_1 , S_2 , and C_2 . For tree #2, R is a qualifying leaf if we disclose C_1 . There is nothing we can do in tree #1 or #3.

Client discloses C_1 .

Stage 7: Server

The server discloses R .

Note: If an omniscient third party combines the policy graphs of the client and the server, they would get the following derivation tree for R .



A safe credential disclosure sequence terminated by R 's disclosure: S_2, C_2, S_1, C_1, R .

Interoperable Strategies in Automated Trust Negotiation

Abstract

Automated trust negotiation is an approach to establishing trust between strangers through the exchange of digital credentials and the use of access control policies that specify what combinations of credentials a stranger must disclose in order to gain access to each local service or credential. We introduce the concept of a trust negotiation *protocol*, which defines the ordering of messages and the type of information messages will contain. To carry out trust negotiation, a party pairs its negotiation protocol with a trust negotiation *strategy* that controls the exact content of the messages, i.e., which credentials to disclose, when to disclose them, and when to terminate a negotiation. There are a huge number of possible strategies for negotiating trust, each with different properties with respect to speed of negotiations and caution in giving out credentials and policies. In the autonomous world of the Internet, entities will want the freedom to choose negotiation strategies that meet their own goals, which means that two strangers who negotiate trust will often not use the same strategy. To date, only a tiny fraction of the space of possible negotiation strategies has been explored, and no two of the strategies proposed so far will interoperate. In this paper, we define a large set of strategies called the *disclosure tree strategy (DTS) family*. Then we prove that if two parties each choose strategies from the DTS family, then they will be able to negotiate trust as well as if they were both using the same strategy. Further, they can change strategies at any point during negotiation. We also show that the DTS family is closed, i.e., any strategy that can interoperate with every strategy in the DTS family must also be a member of the DTS family. We also give examples of practical strategies that belong to the DTS family and fit within the TrustBuilder architecture and protocol for trust negotiation.

1 Introduction

With billions of users on the Internet, most interactions will occur between strangers, i.e., entities that have no pre-existing relationship and may not share a common security domain. In order for strangers to conduct secure transactions, a sufficient level of mutual trust must be established. For this purpose, the *identity* of the participants (e.g., their social security number, fingerprint, institutional tax ID) will often be irrelevant to determining whether or not they should be trusted. Instead, the *properties* of the participants, e.g., employment status, citizenship, group membership, will be most relevant. Traditional security approaches based on identity require a new client to pre-register with the service, in order to obtain a local login, capability, or credential before requesting service; but the same problem arises when the client needs to prove on-line that she is eligible to register with the service. E-commerce needs a more scalable approach that allows automatic on-line pre-registration, or does away entirely with the need for pre-registration. We believe that automated trust establishment is such a solution.

With automated trust establishment, strangers establish trust by exchanging *digital credentials*, the on-line analogues of paper credentials that people carry in their wallets: digitally signed assertions by a credential issuer about the credential owner. A credential is signed using the issuer's

private key and can be verified using the issuer’s public key. A credential describes one or more attributes of the owner, using attribute name/value pairs to describe properties of the owner asserted by the issuer. Each credential also contains the public key of the credential owner. The owner can use the corresponding private key to answer challenges or otherwise demonstrate ownership of the credential. Digital credentials can be implemented using, for example, X.509 [10] certificates.

While some resources are freely accessible to all, many require protection from unauthorized access. Access control policies can be used for a wide variety of “protected” resources, such as services accessed through URLs, roles in role-based access control systems, and capabilities in capability-based systems. Since digital credentials themselves can contain sensitive information, their disclosure will often also be governed by access control policies. For example, suppose that a landscape designer wishes to order plants from Champaign Prairie Nursery (CPN). She fills out an order form on the web, checking an order form box to indicate that she wishes to be exempt from sales tax. Upon receipt of the order, CPN will want to see a valid credit card or her account credential issued by CPN, and a current reseller’s license. The designer has no account with CPN, but she does have a digital credit card. She is willing to show her reseller’s license to anyone, but she will only show her credit card to members of the Better Business Bureau. Therefore, when protected credentials are involved, a more complex procedure needs to be adopted to establish trust through negotiation.

2 Related Work

Credential-based authentication and authorization systems fall into three groups: identity-based, property-based, and capability-based. Originally, public key certificates, such as X.509 [10] and PGP [17], simply bound keys to names, and X.509 v.3 certificates later extended this binding to general properties (attributes). Such certificates form the foundation of identity-based systems, which authenticate an entity’s identity or name and use it as the basis for authorization. Identity is not a useful basis for our aim of establishing trust among strangers.

Systems have emerged that use property-based credentials to manage trust in decentralized, distributed systems [8, 12, 15]. Johnson et al. [12] use attribute certificates (property-based credentials) and use-condition certificates (policy assertions) for access control. Use-condition certificates enable multiple, distributed stakeholders to share control over access to resources. In their architecture, the policy evaluation engine retrieves the certificates associated with a user to determine if the use conditions are met. Their work could use our approach to protect sensitive certificates.

The Trust Establishment Project at the IBM Haifa Research Laboratory [8] has developed a system for establishing trust between strangers according to policies that specify constraints on the contents of public-key certificates. Servers can use a collector to gather supporting credentials from issuer sites. Each credential contains a reference to the site associated with the issuer. That site serves as the starting point for a collector-controlled search for relevant supporting credentials. Security agents in our work could adopt the collector feature, and we could use their policy definition language. Their work could use our approach to protect sensitive credentials and gradually establish trust.

Capability-based systems manage delegation of authority for a particular application. Capability-based systems are not designed for establishing trust between strangers, since clients are assumed to possess credentials that represent authorization of specific actions with the application server. In the capability-based KeyNote system of Blaze et al. [2, 3], a credential describes the conditions under which one principal authorizes actions requested by other principals. KeyNote policies delegate authority on behalf of the associated application to otherwise untrusted parties. KeyNote

credentials express delegation in terms of actions that are relevant to a given application. KeyNote policies do not interpret the meaning of credentials for the application. This is unlike policies designed for use with property-based credentials, which typically derive roles from credential attributes. The IETF Simple Public Key Infrastructure [9] uses a similar approach to that of KeyNote by embedding authorization directly in certificates.

Bonatti et al. [4] introduced a uniform framework and model to regulate service access and information release over the Internet. Their framework is composed of a language with formal semantics and a policy filtering mechanism. Our work can be integrated with their framework.

The P3P standard [14] focuses on negotiating the disclosure of a user's sensitive private information based on the privacy practices of the server. Trust negotiation is generalized to base disclosure on any server property of interest to the client that can be represented in a credential. The work on trust negotiation focuses on certified properties of the credential holder while P3P is based on data submitted by the client that are claims the client makes about itself. Support for both kinds of information in trust negotiation is warranted.

SSL [7], the predominant credential-exchange mechanism in use on the web, and its successor TLS [5, 6] support credentials exchange during client and server authentication. The protocol is suited for identity-based credentials and would need extension to make it adaptable to property-based credentials. Needed additions include protection for sensitive server credentials and a way for the client to explain its policies to the server.

Islam et al. [11] show how to control downloaded executable content using policy graphs. Their system assumes that all the appropriate credentials accompany requests for downloaded content. Their work could be extended using our approach to disclose policies and conduct negotiations.

The first trust negotiation strategies proposed included a naive strategy that discloses credentials as soon as they are unlocked and discloses no policy information, as well as a strategy that discloses credentials only after each party determines that trust can be established, based on reviewing the other party's policies [15]. Yu et al. [16] introduced a new strategy that would succeed whenever success was possible and had certain efficiency guarantees. In [13], consideration was given for sensitive policy information in several strategies that established trust gradually through the introduction of policy graphs. The fact that none of the strategies proposed in this earlier work will interoperate demonstrates the need for trust negotiation protocols and strategy families to support interoperability between negotiation strategies.

3 Trust Negotiation

In our approach to automated trust establishment, trust is established incrementally by exchanging credentials and requests for credentials, an iterative process known as *trust negotiation*. While a *trust negotiation protocol* defines the ordering of messages and the type of information messages will contain, a *trust negotiation strategy* controls the exact content of the messages, i.e., which credentials to disclose, when to disclose them, and when to terminate a negotiation. Figure 1 introduces our *TrustBuilder* architecture for trust negotiation. Each participant in the negotiation has an associated security agent (SA) that manages the negotiation. The security agent mediates access to local protected *resources*, i.e., services and credentials. We say a credential or access control policy is *disclosed* if it has been sent to the other party in the negotiation, and that a service is disclosed if the other party is given access to it. Disclosure of protected resources is governed by access control policies. During a negotiation, the security agent uses a local negotiation strategy to determine what local resources to disclose next, and to accept new disclosures from the other party.

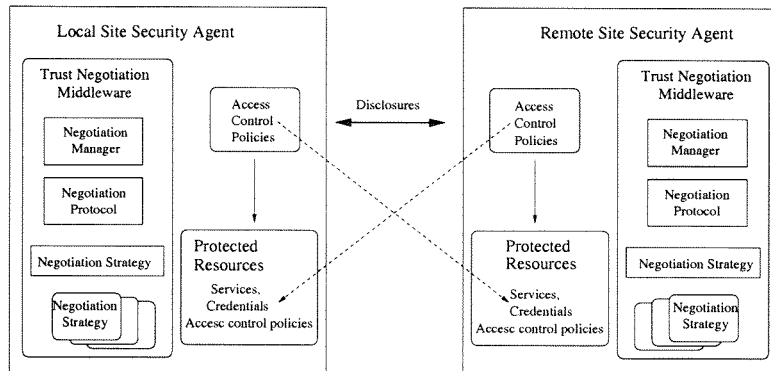


Figure 1: An architecture for automated trust negotiation. A security agent that manages local protected resources and their associated access control policies represents each negotiation participant. A access control policy specifies what resources the other party needs to disclose in order to gain access to a local resource, as indicated by the dotted lines in the figure. Trust negotiation middleware enables negotiation strategy interoperability.

The architecture in figure 1 supports a single protocol for establishing trust, and assumes there will be a variety of negotiation strategies that must be supported. All trust negotiation strategies share the goal of building trust through an exchange of digital credentials that leads to obtaining access to a protected resource. Once enough trust has been established that a particular credential can be disclosed to the other party, a local negotiation strategy must determine whether the credential is relevant to the current stage of the negotiation. Different negotiation strategies will use different definitions of relevance, involving tradeoffs between computational cost, the length of the negotiation, and the number of disclosures.

From the handful of trust negotiation strategies proposed so far in the literature, it is clear that there are endless possible variations in how to negotiate trust. Rather than exploring the space of all possible strategies one strategy at a time, our goal in this paper is to characterize a broad class of strategies (section 6) and design a strategy-independent, language-independent trust negotiation protocol (section 5) that ensures their interoperability within the TrustBuilder trust negotiation architecture.

4 Access Control Policies

We assume that the information contained in access control policies (*policies*, for short) and credentials can be expressed as finite sets of statements in a formal language with a well-defined semantics. XML or logic programming languages with appropriate semantics may be suitable languages in practice [8, 1]. For convenience, we will assume that the original language allows us to describe the meaning of a set of statements as the set of all models that satisfy the set of statements, in the usual logic sense. We say that a set X of statements *satisfies* a set of statements P if and only if P is true in all models of X . For purely practical reasons, we require that the language be *monotonic*, i.e., if a set of statements X satisfies policy P , then any superset of X will also satisfy P ; that way, once a negotiation strategy has determined that the credentials disclosed by a participant satisfy the policy of a resource, the strategy knows that the same policy will be satisfied for the rest of the negotiation, and does not have to be rechecked.

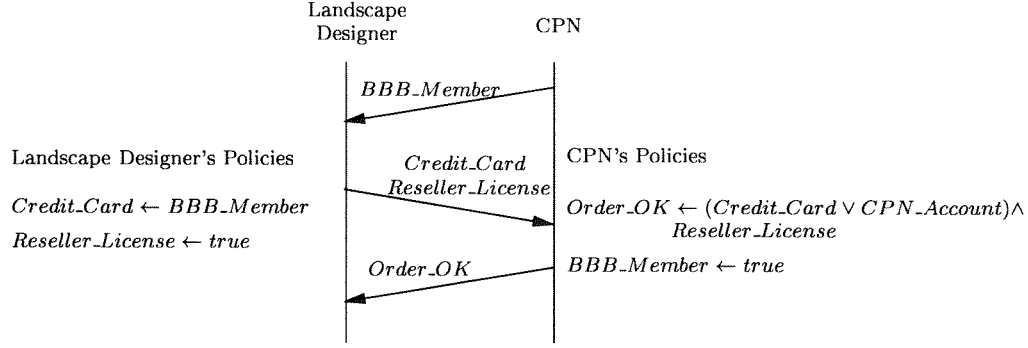


Figure 2: An example of access control policies and a safe disclosure sequence which establishes trust between the server and the client.

In this paper, we will treat credentials and services as propositional symbols. Each of these resources has exactly one access control policy, of the form $C \leftarrow F_C(C_1, \dots, C_k)$, where $F_C(C_1, \dots, C_k)$ is a Boolean expression involving only credentials $C_1, \dots, C_k$ that the other party may possess, Boolean constants *true* and *false*, the Boolean operators $\vee$ and $\wedge$, and parentheses as needed. C_i is satisfied if and only if the other party has disclosed credential C_i . We assume that we can distinguish between local and remote resources (by renaming propositional symbols as necessary). Resource C is *unlocked* if its access control policy is satisfied by the set of credentials disclosed by the other party. A resource is *unprotected* if its policy is always satisfied. The *denial policy* $C \leftarrow \text{false}$ means that either the party does not possess C , or else will not disclose C under any circumstances. A party implicitly has a denial policy for each credential it does not possess. If the disclosure of a set S of credentials satisfies resource R 's policy, then we say S is a *solution set* for R . Further, if none of S 's proper subsets is a solution set for R , we say S is a *minimal solution set* for R . The *size* of a policy is the number of symbol occurrences in it.

Given sequence $G = (C_1, \dots, C_n)$ of disclosures of protected resources, if each C_i is unlocked at the time it is disclosed, $1 \leq i \leq n$, then we say G is a *safe disclosure sequence*. The goal of trust negotiation is to find a safe disclosure sequence $G = (C_1, \dots, C_n = R)$, where R is the resource to which access was originally requested. When this happens, we say that trust negotiation succeeds. If $C_i = C_j$ and $1 \leq i < j \leq n$, then we say G is *redundant*. Since the language used to represent policies and credentials is monotonic, we can remove the later duplicates from a redundant safe disclosure sequence and the resulting sequence is still safe. Figure 2 shows a safe disclosure sequence for the landscape designer's purchase from CPN discussed earlier. A more complex example can be found in Appendix B. It is important to note that this example, and our algorithms that follow, rely on *lower levels of software* to perform the functions associated with disclosure of a credential: verification of its contents, checks for revocation as desired, checks of validity dates, authentication of ownership, etc., as is normally done for X.509 certificates. The necessary calls to these functions, and the translation of credentials into the language used to represent policies, are not shown in our algorithms.

5 The TrustBuilder Protocol and Strategy Families

Previous work on trust negotiation has not explicitly proposed any trust negotiation protocols, instead defining protocols implicitly by the way each negotiation strategy works. This is one reason

```

TrustBuilder_handle_disclosure_message ( $m, R$ )
Input:  $m$  is the last disclosure message received from the remote party.
       $R$  is the resource to which the client originally requested access.
TrustBuilder_check_for_termination( $m, R$ ). //Stop negotiating, if appropriate.
TrustBuilder_next_message( $m, R$ ).
End of TrustBuilder_handle_disclosure_message.

TrustBuilder_next_message( $m, R$ )
// First, let the local strategy suggest what the next message should be.
Let  $G$  be the disclosure message sequence so far.
Let  $L$  be the local resources and policies.
 $S_m = \text{Local\_strategy}(G, L, R)$ .
//  $S_m$  contains the candidate messages the local strategy suggests.
Choose any single message  $m'$  from  $S_m$ .
Send  $m'$  to the remote party.
TrustBuilder_check_for_termination( $m', R$ ). //Stop negotiating, if appropriate.
End of TrustBuilder_next_message.

TrustBuilder_check_for_termination( $m, R$ )
If  $m$  is the empty set  $\emptyset$  and this is not the beginning of the negotiation,
    Then negotiations have failed. Stop negotiating and exit.
If  $m$  contains the disclosure of  $R$ ,
    Then negotiations have succeeded. Stop negotiating and exit.
End of TrustBuilder_check_for_termination.

```

Figure 3: Pseudocode for the TrustBuilder protocol. The negotiation is triggered when the client asks to access a protected resource owned by the server. After rounds of disclosures, either one party sends a failure message and ends the negotiation, or the server grants the client access.

why no two different previously proposed strategies can interoperate – their underlying protocols are totally different.

We remedy this problem by defining a simple protocol for TrustBuilder. Formally, a *message* in the TrustBuilder protocol is a set $\{R_1, \dots, R_k\}$ where each R_i is a disclosure of a local credential, a local policy, or a local resource. When a message is the empty set $\emptyset$, we also call it a *failure message*. Further, to guarantee the safety and timely termination of trust negotiation no matter what policies and credentials the parties possess, the TrustBuilder protocol requires the negotiation strategies used with it to enforce the following three conditions throughout negotiations:

1. If a message contains a denial policy disclosure $C \leftarrow \text{false}$, then C must appear in a previously disclosed policy.
2. A credential or policy can be disclosed at most once.
3. Every disclosure must be safe.

Before the negotiation starts, the client sends the original resource request message to the server indicating its request to access resource R . This request triggers the negotiation, and the server invokes its local security agent with the call `TrustBuilder_handle_disclosure_message( $\emptyset, R$ )`. Then the client and server exchange messages until either the service R is disclosed by the server or one party sends a failure message. The whole negotiation process is shown in figure 3.

In the remainder of this paper, unless otherwise noted, we discuss only strategies that can be called from the TrustBuilder protocol and satisfy the three conditions above. A formal definition of a negotiation strategy is given below.

Definition 5.1. A strategy is a function $f : \{G, L, R\} \rightarrow S_m$, where R is the resource to which the client originally requested access, $G = (m_1, \dots, m_k)$ is a sequence of disclosure messages such that $m_i \neq \emptyset$ and $R \notin m_i$ for $1 \leq i \leq k$, L is the set of local resources and policies, and S_m is a set of disclosure messages. Further, every disclosure in a message in S_m must be of a local resource or policy, as must be all the disclosures in m_{k-2i} , for $1 \leq k-2i < k$. The remaining disclosures in G are of remote resources and policies.

Intuitively, based on the disclosures made so far, plus the local resources and policies, a negotiation strategy will suggest the next set of disclosures to send to the other party. Note that a strategy returns a *set* of possible disclosure messages, rather than a single message. Practical negotiation strategies will suggest a single next message, but the ability to suggest several possible next messages will be very convenient in our formal analysis of strategy properties, so we include it both in the formal definition of a negotiation strategies and also in the protocol pseudocode in figure 3.

Definition 5.2. Strategies f_A and f_B are compatible if whenever there exists a safe disclosure sequence for a party $\mathcal{P}_A$ to obtain access to a resource owned by party $\mathcal{P}_B$, the trust negotiation will succeed when $\mathcal{P}_A$ uses f_A and $\mathcal{P}_B$ uses f_B . If $f_A = f_B$, then we say that f_A is self-compatible.

Definition 5.3. A strategy family is a set $\mathcal{F}$ of mutually compatible strategies, i.e., $\forall f_1 \in \mathcal{F}, f_2 \in \mathcal{F}$, f_1 and f_2 are compatible. We say a set $\mathcal{F}$ of strategies is closed if given a strategy f' , if f' is compatible with every strategy in $\mathcal{F}$, then $f' \in \mathcal{F}$.

One obvious advantage of strategy families is that a security agent (SA) can choose strategies based on its needs without worrying about interoperability, as long as it negotiates with other SAs that use strategies from the same family. As another advantage, under certain conditions, an SA does not need to stick to a fixed strategy during the entire negotiation process. It can adopt different strategies from the family in different phases of the negotiation. For example, during the early phase, since the trust between two parties is very limited, an SA may adopt a cautious strategy for disclosing credentials. When a certain level of trust has been established, in order to accelerate the negotiation, the SA may adopt a less cautious strategy. However, without the closure property, a family may not be large enough for practical use. As an extreme example, given any self-compatible strategy f , $\{f\}$ is a strategy family. The closure property guarantees the maximality of a strategy family.

In general, the notions of strategy families and closed sets of strategies are incomparable, in the sense that neither of them implies the other. For example, if a strategy f 's output is $\{m\}$, where m is a message containing all the undisclosed local policies and unlocked credentials, then it is easy to prove that f is self-compatible. Then $\{f\}$ is a family, but by no means closed. On the other hand, consider the strategy f' whose output is always $\{\emptyset\}$. Obviously f' is not compatible with any strategies. The strategy set $\{f'\}$ is not a family but is closed.

We end this section with two simple propositions.

Proposition 5.1. Any subset of a strategy family is also a family.

Proposition 5.2. If a strategy family $\mathcal{F}$ is a proper subset of another family, then $\mathcal{F}$ is not closed.

6 Characterizing Safe Disclosure Sequences

In this section, we define the concepts that we use to describe the progress of a negotiation and to characterize the behavior of different strategies. In the remainder of the paper, we use R to represent the resource to which access was originally requested.

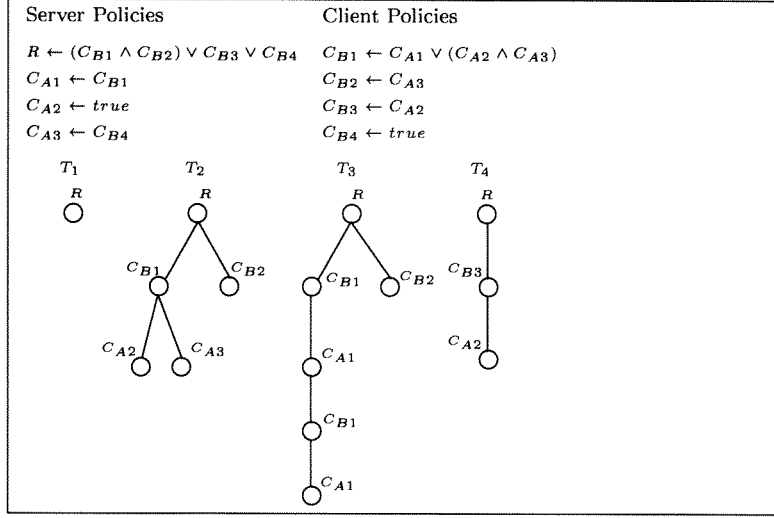


Figure 4: Example disclosure trees for a set of policies

6.1 Disclosure Trees

Definition 6.1. A disclosure tree for R is a finite tree satisfying the following conditions:

1. The root represents R .
2. Except for the root, each node represents a credential. When the context is clear, we refer to a node by the name of the credential it represents.
3. The children of a node C form a minimal solution set for C .

When all the leaves of a disclosure tree T are unprotected credentials, we say T is a full disclosure tree. Given a disclosure tree T , if there is a credential appearing twice in the path from a leaf node to the root, then we call T a redundant disclosure tree.

Figure 4 shows example disclosure trees. Note that T_2 is redundant and T_4 is a full disclosure tree.

The following theorems state the relationship between disclosure trees and safe disclosure sequences that lead to the granting of access to resource R . Proofs of all theorems can be found in Appendix A.

Theorem 6.1. Given a non-redundant safe disclosure sequence $G = (C_1, \dots, C_n = R)$, there is a full non-redundant disclosure tree T such that both of the following hold:

1. The nodes of T are a subset of $\{C_1, \dots, C_n\}$.
2. For all credential pairs (C'_1, C'_2) such that C'_1 is an ancestor of C'_2 in T , C'_2 is disclosed before C'_1 in G . □

Theorem 6.2. Given a full disclosure tree for R , there is a non-redundant safe disclosure sequence ending with the disclosure of R . □

By theorems 6.1 and 6.2, we get the following corollary immediately.

Corollary 6.1. *Given a safe disclosure sequence $G = (C_1, \dots, C_n = R)$, there is a full non-redundant disclosure tree T such that:*

1. *The credential nodes of T are a subset of $\{C_1, \dots, C_n\}$,*
2. *For all credential pairs (C'_1, C'_2) such that C'_1 is an ancestor of C'_2 in T , the first disclosure of C'_2 in G is before the first disclosure of C'_1 .*

Without loss of generality, from now on, we consider only non-redundant disclosure sequences.

Since there is a natural mapping between safe disclosure sequences and disclosure trees, during the negotiation, theoretically one could determine whether a potential credential or policy disclosure is helpful by examining all the disclosure trees for R . At the beginning of a negotiation, before disclosures begin, the only relevant disclosure tree for the client contains a single node R . As the negotiation proceeds, other trees may become relevant. The following definitions help us describe the set of relevant trees.

Definition 6.2. *Given a disclosure tree T and a set of S_c of credentials, the reduction of T by S_c , $\text{reduction}(T, S_c)$, is the disclosure tree T' which is obtained by removing all the subtrees rooted at a node representing resource $C \in S_c$. Given a set S_t of disclosure trees, $\text{reduction}(S_t, S_c) = \{\text{reduction}(T, S_c) \mid T \in S_t\}$.*

If S_c is the set of credential disclosures made so far, then reducing T by S_c prunes out the part of the negotiation that has already succeeded. Intuitively, if a credential C has been disclosed, then we already have a safe disclosure sequence for C . We do not need to disclose additional credentials or policies in order to get a full disclosure tree rooted at C . An example of a disclosure tree reduction is shown in figure 5(a).

Definition 6.3. *Given a disclosure tree T and a policy set S_p containing no denial policies, the expansion of T by S_p , $\text{expansion}(T, S_p)$, is the set of all disclosure trees T_i such that*

1. *T is a subgraph of T_i , i.e., there exists a set S of credentials such that $\text{reduction}(T_i, S) = T$.*
2. *For each edge (C_1, C_2) in T_i , if (C_1, C_2) is not an edge of T , then C_1 's policy is in S_p .*
3. *For each leaf node C of T_i , either S_p does not contain C 's policy, or T_i is redundant.*

Given a set of disclosure trees S_t , $\text{expansion}(S_t, S_p) = \bigcup_{T \in S_t} \text{expansion}(T, S_p)$.

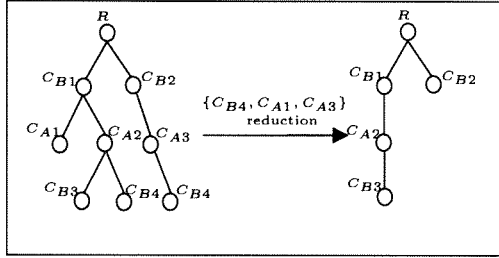
A disclosure tree can expand when a party receives new policy disclosures. An example of a disclosure tree expansion is shown in figure 5(b).

Definition 6.4. *Given a set S_t of disclosure trees and a set S_{dp} of denial policies, the denial pruning of S_t by S_{dp} , denoted $\text{prune}_{\text{denial}}(S_t, S_{dp})$, is the set*

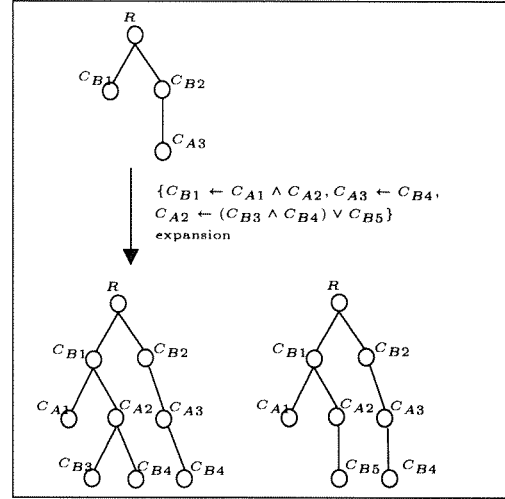
$$\{T \mid T \in S_t \text{ and } T \text{ contains no resource whose policy is in } S_{dp}\}.$$

Since a full disclosure tree contains only credentials that the two parties possess, if a disclosure tree node represents a credential with a denial policy, that tree cannot evolve into a full disclosure tree, and is no longer relevant.

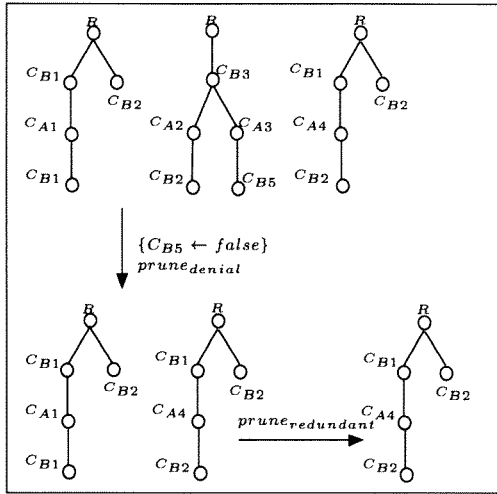
Definition 6.5. *Given a set S_t of disclosure trees, the redundancy pruning of S_t , denoted $\text{prune}_{\text{redundant}}(S_t)$, is the set*



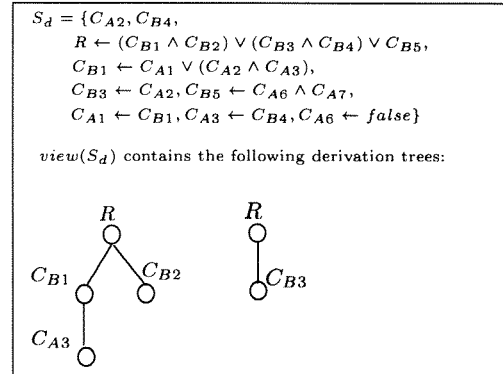
(a) Example of a disclosure tree reduction



(b) Example of a disclosure tree expansion



(c) Example of denial pruning and redundancy pruning



(d) Example of a view

Figure 5: Examples of operations on disclosure trees

$$\{T \mid T \in S_t \text{ and } T \text{ is not a redundant disclosure tree}\}.$$

The rationale for redundancy pruning will be shown after we introduce more operations on disclosure trees. Examples of denial and redundancy pruning are shown in figure 5(c).

Definition 6.6. Given a disclosure tree T and a set S_{dp} of denial policies, S_p of non-denial policies, and S_c of credentials, let $S = S_{dp} \cup S_p \cup S_c$. The evolution of T by S , denoted $\text{evolution}(T, S)$, is

$$\text{prune}_{\text{redundant}}(\text{prune}_{\text{denial}}(\text{reduction}(\text{expansion}(T, S_p), S_c), S_{dp})).$$

Given a set S_t of disclosure trees, $\text{evolution}(S_t, S) = \bigcup_{T \in S_t} \text{evolution}(T, S)$. As a special case, when T is the disclosure tree containing only a root node R , then we say $\text{evolution}(T, S)$ is the view of S , denoted $\text{view}(S)$.

During the negotiation, let S be the set of credentials and policies disclosed so far and L be the local policies of a negotiation party. Then $\text{view}(S \cup L)$ contains all the relevant disclosure trees which can be seen by this party. An example view is shown in figure 5(d). Sometimes even though a tree may evolve into a full tree later in the negotiation, it is nonetheless redundant and can be removed by redundancy pruning, whose correctness is guaranteed by the following theorem.

Theorem 6.3. Let T be a full but redundant disclosure tree. Then there is a full disclosure tree T' that is not redundant. $\square$

Suppose S is the set of currently disclosed credentials and policies. By theorem 6.3, if a redundant tree may evolve into a full tree, then the corresponding non-redundant tree is already included in $\text{view}(S)$. So the redundant trees are not relevant for the remainder of the negotiation.

In order to make a negotiation successful whenever the policies of the two negotiation parties allow, the negotiation strategy should make sure no possible full disclosure trees have been overlooked. A disclosure tree also tells a party what may contribute to the success of a negotiation. As an example, suppose party $\mathcal{P}_B$ requests service R from party $\mathcal{P}_A$. S_d , the set of disclosures so far, and $\text{view}(S_d)$ are shown in figure 6. Suppose now it is $\mathcal{P}_A$'s turn to send a message to $\mathcal{P}_B$. From the disclosure tree, it is clear to an outside observer that credentials C_{A1} and C_{A2} must be disclosed if the negotiation is to succeed. So $\mathcal{P}_A$'s negotiation strategy can now disclose C_{A1} 's and/or C_{A2} 's policy. This example shows that in order to let a negotiation party $\mathcal{P}$ know what might be the next appropriate message, a disclosure tree should have at least one leaf node that is a credential that the other party wants $\mathcal{P}$ to disclose. We have the following definition:

Definition 6.7. Disclosure tree T 's evolvable leaves for party $\mathcal{P}_A$, denoted $\text{evolvable}(T, \mathcal{P}_A)$, are the set of leaf nodes C of T such that either $C = R$ and $\mathcal{P}_A$ is the server, or C appears in a policy that $\mathcal{P}_B$ disclosed to $\mathcal{P}_A$. If $\text{evolvable}(T, \mathcal{P}_A) \neq \emptyset$, we say T is evolvable for $\mathcal{P}_A$.

The disclosure tree in figure 6 is evolvable for both $\mathcal{P}_A$ and $\mathcal{P}_B$.

If a negotiation reaches a point where every leaf node of some disclosure tree is unlocked, then the tree is a full tree and corresponds to a safe disclosure sequence.

Definition 6.8. Let $\mathcal{P}_A$ be a negotiation party, T be a disclosure tree, and S be a set of policies and credentials. If every resource in $\text{evolvable}(T, \mathcal{P}_A)$ is unlocked by credentials in S , then we say T is semi-full with S for $\mathcal{P}_A$. Further, we say T is full with S iff every leaf node of T is unlocked by credentials in S .

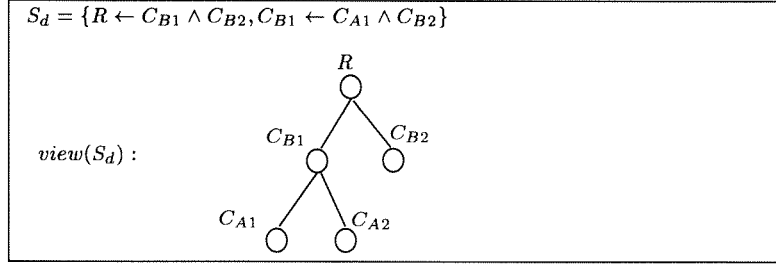


Figure 6: $view(S_d)$ where $S_d = \{R \leftarrow C_{B1} \wedge C_{B2}, C_{B1} \leftarrow C_{A1} \wedge C_{A2}\}$

6.2 Strategy Caution and Strategy Set Generators

If $\mathcal{F}$ is a strategy family, then intuitively, every strategy in $\mathcal{F}$ always discloses enough information to keep the negotiation moving towards success, if success is possible. If $\mathcal{F}$ is also closed, then $\mathcal{F}$ must also contain those strategies that disclose only the minimal amount of information needed to continue negotiations. Therefore it is helpful to formally define a relationship between strategies based on the information they disclose.

Definition 6.9. *Given two negotiation strategies f_1 and f_2 , if for all possible inputs G , L , and R to f_1 and f_2 , we have*

$$\forall m \in f_2(G, L, R) \exists m' \in f_1(G, L, R) \text{ such that } m' \subseteq m$$

then we say f_1 is at least as cautious as f_2 , denoted as $f_1 \preceq f_2$ or $f_2 \succeq f_1$.

Caution defines a partial order between strategies. Intuitively, if $f_2 \succeq f_1$ then f_2 always discloses at least as much information as f_1 does.

Definition 6.10. *Given a strategy f , the set of strategies generated by f , denoted $StraSet(f)$, is the set $\mathcal{F} = \{f' | f' \succeq f\}$. f is called the generator of $\mathcal{F}$.*

As we discussed in section 6.1, during a trust negotiation, evolvable trees give guidance on what a party needs to disclose in the next message so that the whole negotiation advances towards potential success. If there is no evolvable tree, then a cautious party will choose to end the negotiation even if the policies of the two parties allow success. Therefore, to ensure that negotiations succeed whenever possible, a strategy must ensure that the other party will have an evolvable tree when the other party needs to make its next disclosure. The only exception is when the strategy knows that no disclosure tree can evolve into a full tree.

7 The Disclosure Tree Strategy Family

We present the *disclosure tree strategy* (DTS), then prove that DTS generates a closed family. Throughout this section, we assume that $G = (m_1, \dots, m_k)$ is a sequence of messages such that $m_i \neq \emptyset$ and $R \notin m_i$ for $1 \leq i \leq k$. We assume L_A and L_B are the local policies of parties $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively, and $S_d = \bigcup_{1 \leq i \leq k} m_i$. Without loss of generality, we assume $\mathcal{P}_A$ will send the next message to $\mathcal{P}_B$.

Definition 7.1. *The Disclosure Tree Strategy (DTS for short) is a strategy $DTS(G, L_A, R)$ such that:*

- 1) $DTS(G, L_A, R) = \{\emptyset\}$ if and only if $view(S_d \cup L_A) = \emptyset$ or $view(S_d)$ has no evolvable tree for $\mathcal{P}_A$.
- 2) Otherwise, $DTS(G, L_A, R)$ contains all messages m' such that one of the following conditions holds:
 - $m' = \{R\}$, if R is unlocked by credentials in S_d ;
 - m' is a non-empty set of credentials and policies such that $view(S_d \cup m')$ contains at least one evolvable tree for $\mathcal{P}_B$, and no non-empty proper subset of m' has this property.

Condition 1) states under what circumstances the DTS strategy will terminate the negotiation with a failure message. Condition 2) guarantees that the other party will have an evolvable tree. Therefore, the other party can always send a message back that evolves a disclosure tree. Thus, no failure message will be sent unless there is no disclosure tree at all, in which case the negotiation cannot succeed anyway. Formally, we have the following theorems:

Theorem 7.1. *The set of strategies generated by DTS is a family.* □

Theorem 7.2. *If a strategy f and DTS are compatible, then $f \succeq DTS$.* □

We call the family generated by DTS the *DTS family*. By theorems 7.1 and 7.2, we get the following corollary immediately.

Corollary 7.1. *The DTS family is closed.* □

As we mentioned in section 5, one advantage of a strategy family can be the ability to adopt different strategies from a family in different phases of the negotiation. Correct interoperability is guaranteed as long as both parties' strategies are from the same family.

Definition 7.2. *Let f_1 and f_2 be two strategies. A strategy f' is a hybrid of f_1 and f_2 if $\forall G, L, R$, $f'(G, L, R) \subseteq f_1(G, L, R) \cup f_2(G, L, R)$ and $f' \neq f_1$ and $f' \neq f_2$.*

If a security agent adopts different DTS family strategies in different phases of trust negotiation, it is equivalent to adopting a hybrid of those strategies.

Theorem 7.3. *Let f_1 and f_2 be strategies in the DTS family and let f' be a hybrid of f_1 and f_2 . Then f' is also in the DTS family.* □

Therefore, as long as both parties use strategies from the DTS family, they can switch between different practical strategies as often as they like, and trust negotiation will still succeed whenever possible.

Although disclosure trees are a useful tool for understanding strategy properties, it would require exponential time and space to materialize all the disclosure trees during a negotiation. Fortunately, many strategies in the DTS family are quite efficient. We present two efficient strategies: TrustBuilder-Simple and TrustBuilder-Relevant, which are both in the DTS family.

The TrustBuilder-Simple strategy puts all undisclosed policies and unlocked credentials in the next message to the other party. If all the policies and unlocked credentials have already been disclosed, it will send a failure message. Its pseudocode is shown in figure 7(a).

We say a credential C is *syntactically relevant* to resource R iff C appears in R 's policy, or C appears in the policy of a credential C' that is relevant to R . In contrast to TrustBuilder-Simple, the TrustBuilder-Relevant strategy (figure 7(b)) discloses a credential C 's policy only if C is syntactically relevant to R . Similarly, TrustBuilder-Relevant only discloses syntactically relevant unlocked credentials.

TrustBuilder-Simple strategy*Input:* $G = (m_1, \dots, m_k)$: a sequence of safe disclosure messages. L : the local resources and policies of this party. R : the resource to which access was originally requested.*Output:*A set containing a single disclosure message m .*Pre-condition:* R has not been disclosed and $m_k \neq \emptyset$.Let $\mathcal{P}_A$ be the local party and $\mathcal{P}_B$ the remote party. $S_d = \bigcup_{1 \leq i \leq k} m_i$; $m = \emptyset$;For every local credential C that is unlocked by S_d $m = m \cup \{C\}$;For every local locked credential C if (C 's policy P is not a denial policy)then $m = m \cup \{P\}$;For every policy $P' \in S_d$ such that $P' \notin L$ For every credential that C appears in P' and has a denial policy $m = m \cup \{C \leftarrow false\}$; $m = m - S_d$;return $\{m\}$;

(a) Pseudocode for the TrustBuilder-Simple strategy

TrustBuilder-Relevant-Strategy*Input:* $G = (m_1, \dots, m_k)$: a sequence of safe disclosure messages. L : the local resources and policies of this party. R : the resource to which access was originally requested.*Output:*A set containing a single disclosure message m .*Pre-condition:* R has not been disclosed and $m_k \neq \emptyset$.Let $\mathcal{P}_A$ be the local party and $\mathcal{P}_B$ the remote party. $S_d = \bigcup_{1 \leq i \leq k} m_i$; $m = \emptyset$;For every local credential C syntactically relevant to R if (C is unlocked by S_d)then $m = m \cup \{C\}$;else $m = m \cup \{C's\ policy\}$;For every policy $P' \in S_d$ such that $P' \notin L$ For every credential C that appears in P' and has a denial policy $m = m \cup \{C \leftarrow false\}$; $m = m - S_d$;return $\{m\}$;

(b) Pseudocode for the TrustBuilder-Relevant strategy

Figure 7: Pseudocode for two strategies in the DTS family

Proposition 7.1. *If a credential C appears in a disclosure tree for R , then C is relevant to R .*

Theorem 7.4. *TrustBuilder-Simple and TrustBuilder-Relevant belong to the DTS family.* $\square$

Theorem 7.5. *The computation costs of TrustBuilder-Simple and TrustBuilder-Relevant in the whole process of trust negotiation are bounded by $O(nm)$, where n is the total number of credentials and m is the total size of the policies of both parties.* $\square$

The worst-case behavior of TrustBuilder-Simple and TrustBuilder-Relevant occurs when every credential belonging to one party appears in every policy belonging to the other party, and each disclosure message discloses a single credential or policy.

8 Summary and Future Work

Instead of proposing another strategy for automated trust negotiation, this paper focuses on guaranteeing interoperability between different strategies. We first propose a very simple trust negotiation protocol for the TrustBuilder trust negotiation architecture. Then we study strategies that adhere to this protocol. We introduce the concepts of strategy families and closed sets of strategies. If two strategies are in the same strategy family, then they will always correctly interoperate with each other. Closure expresses the maximality of a strategy family, i.e., if we add another strategy to a closed family, the resulting set of strategies is no longer a family. In practice, we want to identify closed families of strategies because they give negotiation participants maximum freedom in choosing the strategies appropriate for them. We introduce the concept of disclosure trees and identify the natural mapping between full disclosure trees and safe credential disclosure sequences. We then propose a strategy called the disclosure tree strategy (DTS), and prove that all the strategies that are no more cautious than DTS form a closed strategy family. Finally we give examples of practical strategies from the DTS family.

In this paper, we assume a credential's policy is freely available, which means it can be shown to others whenever requested. However, some policies contain sensitive information that should be protected from arbitrary disclosure. We are currently investigating strategy families for use in this situation and with non-propositional policy languages. We are also implementing TrustBuilder for testbed experimentation in e-commerce applications, and investigating more sophisticated definitions of "minimal" disclosure for use with practical policies.

References

- [1] K. R. Apt, D. S. Warren, and M. Truszczynski (editor). *The Logic Programming Paradigm: A 25-Year Perspective*. Springer-Verlag, 1999.
- [2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System Version 2. In *Internet Draft RFC 2704*, September 1999.
- [3] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. In *Security Protocols Workshop*, Cambridge, UK, 1998.
- [4] P. Bonatti and P. Samarati. Regulating Service Access and Information Release on the Web. In *Conference on Computer and Communications Security*, Athens, November 2000.
- [5] T. Dierks and C. Allen. The TLS Protocol Version 1.0. In <http://www.ietf.org/rfc/rfc2246.txt>, January 1999.

- [6] S. Farrell. TLS Extension for Attribute Certificate Based Authorization. In <http://www.ietf.org/internet-drafts/draft-ietf-tls-attr-cert-01.txt>, August 1998.
- [7] A. Frier, P. Karlton, and P. Kocher. *The SSL 3.0 Protocol*. Netscape Communications Corp., November 1996.
- [8] A. Herzberg, J. Mihaeli, Y. Mass, D. Naor, and Y. Ravid. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.
- [9] <http://www.ietf.org/html.charters/spki-charter.html>. *Simple Public Key Infrastructure (SPKI)*.
- [10] International Telecommunication Union. *Rec. X.509 - Information Technology - Open Systems Interconnection - The Directory: Authentication Framework*, August 1997.
- [11] N. Islam, R. Anand, T. Jaeger, and J. R. Rao. A Flexible Security System for Using Internet Content. *IEEE Software*, 14(5), September 1997.
- [12] W. Johnson, S. Mudumbai, and M. Thompson. Authorization and Attribute Certificates for Widely Distributed Access Control. In *IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1998.
- [13] K. Seamons, M. Winslett, and T. Yu. Limiting the Disclosure of Access Control Policies during Automated Trust Negotiation. In *Network and Distributed System Security Symposium*, San Diego, CA, April 2001.
- [14] W3C, <http://www.w3.org/TR/WD-P3P/Overview.html>. *Platform for Privacy Preferences (P3P) Specification*.
- [15] W. Winsborough, K. Seamons, and V. Jones. Negotiating Disclosure of Sensitive Credentials. In *Second Conference on Security in Communication Networks*, Amalfi, Italy, September 1999.
- [16] T. Yu, X. Ma, and M. Winslett. PRUNES: An Efficient and Complete Strategy for Automated Trust Negotiation over the Internet. In *Conference on Computer and Communication Security*, Athens, Greece, November 2000.
- [17] P. Zimmerman. *PGP User's Guide*. MIT Press, 1994.

The appendix is not part of the paper, and is only provided to aid the reviewers.

A Proofs of Theorems

Proof of theorem 6.1: By induction on n .

When $n = 1$, resource R is unprotected. We can have a disclosure tree with only the root node, and the theorem holds.

Assume when $n \geq k \geq 1$, the theorem holds.

When $n = k + 1$, since resource R is eventually granted, we can find a minimal solution set $\{C_{j_1}, \dots, C_{j_t}\}$ for R among the credentials in G . Since G is a non-redundant safe disclosure sequence, for each C_{j_i} , where $1 \leq i \leq t$, $(C_1, \dots, C_{j_i})$ is a safe disclosure sequence with length no more than k . By the induction hypothesis, there is a non-redundant disclosure tree T_i whose root is C_{j_i} and all the nodes are credentials appearing in $(C_1, \dots, C_{j_i})$, which is a prefix of G . And for every credential pair (C'_1, C'_2) such that C'_1 is an ancestor of C'_2 in T_i , C'_2 is disclosed before C'_1 in $(C_1, \dots, C_{j_i})$. By adding R as the root and all the roots of T_i , $1 \leq i \leq t$, as children of R , we get a full non-redundant disclosure tree that satisfies conditions 1) and 2) in the theorem. Therefore the theorem holds. $\square$

Proof of theorem 6.2: Let $G = (C_1, \dots, C_n = R)$ be the post-order traversal of T . According to the definition of full disclosure trees, when a credential is disclosed in G , either it is unprotected or one of its minimal solution sets has been disclosed. Therefore its disclosure is safe, and G is a safe disclosure sequence. For every C_i in G , if there exists a credential disclosure C_j such that $j < i$ and $C_j = C_i$ in G , then we remove C_i from G . The resulting disclosure sequence is safe and non-redundant. $\square$

Proof of theorem 6.3: By induction on n , the number of nodes in T . When $n = 1$, with only a root node, T is redundant.

When $n = 2$, because T is a redundant tree, it must have only one edge (R, R) . Since T is a full tree, R is unprotected. So the tree with only the root node R is a non-redundant full disclosure tree, and T is not a legal disclosure tree.

Assume when $n = k$, where $k \geq 2$, the theorem holds.

When $n = k + 1$, let credential C' appear more than once in the path from the root to a leaf node. Suppose the path is $(R, C_1, \dots, C_i = C', \dots, C_j = C', \dots, C_k)$ such that C_i and C_j are the first and last appearance of C' in the path respectively. Since T is a full disclosure tree, the subtree rooted at C_j is also a full tree. We replace the subtree rooted at C_i by the one rooted at C_j . The resulting disclosure tree is still full but with at most k nodes. By the induction hypothesis, there is a full disclosure tree T' that is not redundant. $\square$

Proof of theorem 7.1: Suppose $\mathcal{P}_A$ and $\mathcal{P}_B$ adopt strategies f_1 and f_2 respectively, and f_1 and f_2 belong to $\text{StraSet}(DTS)$. We need to prove that if the negotiation fails, there is no safe disclosure sequence leading to the grant of access to the originally requested resource R .

When the negotiation fails, without loss of generality, we assume it is $\mathcal{P}_A$ that sends the failure message. Suppose that before $\mathcal{P}_A$ sends the failure message, $G = (m_1, \dots, m_k)$ is the sequence of exchanged messages. Therefore m_k is the last message in G that $\mathcal{P}_B$ sent to $\mathcal{P}_A$. Let L_A and L_B be the local policies of $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively, and $S_d = \bigcup_{1 \leq i \leq k} m_i$. Since $\emptyset \in f_1(G, L_A, R)$ and $f_1 \succeq DTS$, we must have $DTS(G, L_A, R) = \{\emptyset\}$. According to definition 7.1, one of the following must be true:

1. $view(S_d \cup L_A) = \emptyset$.
2. $view(S_d)$ has no evolvable tree for $\mathcal{P}_A$.

When 2) holds but 1) does not hold, we must have

$$DTS((m_1, \dots, m_{k-1}), L_B, R) = \{\emptyset\}.$$

Otherwise, since $f_2 \succeq DTS$ and $R \notin m_k$, m_k must be a superset of a minimal set m' of credentials and policies such that $view((S_d - m_k) \cup m')$ has at least one evolvable tree for $\mathcal{P}_A$. Therefore, $view(S_d)$ also has at least one evolvable tree for $\mathcal{P}_A$, which contradicts 2). Since $DTS((m_1, \dots, m_{k-1}), L_B, R) = \{\emptyset\}$, that means that before $\mathcal{P}_B$ sent m_k , one of 1) or 2) was true. Since at the beginning of the negotiation, 2) is not satisfied, we know that in some previous stage of the negotiation, 1) must have been true.

When 1) holds, that means no tree will evolve to be a full disclosure tree. So there is no safe disclosure sequence leading to the grant of access to R . $\square$

Proof of theorem 7.2: By contradiction.

Suppose that $\mathcal{P}_A$ is the local party. Assuming $DTS \not\preceq f$, then there exists a choice of G , L_A and R such that

$$\exists m' \in f(G, L_A, R) \text{ such that } \forall m \in DTS(G, L_A, R), m \not\subseteq m'$$

Then the following must be true:

- $view(S_d \cup L_A) \neq \emptyset$.
- $view(S_d)$ has at least one evolvable tree for $\mathcal{P}_A$.

Otherwise, according to definition 7.1, $DTS(G, L_A, R) = \{\emptyset\}$ and $\emptyset \subseteq m'$. Also, we have $R \notin m'$. Otherwise, R is unlocked by S_d and $\{R\} \in DTS(G, L_A, R)$. $\{R\} \subseteq m'$.

$DTS(G, L_A, R)$ contains all the minimal sets m of local policies and credentials such that $view(S_d \cup m)$ has at least one evolvable tree for $\mathcal{P}_B$. Since $DTS \not\preceq f$, $view(S_d \cup m')$ has no evolvable tree for $\mathcal{P}_B$. Thus, after sending m' , DTS at $\mathcal{P}_B$ will send a failure message and end the negotiation. However, since $view(S_d \cup L_A) \neq \emptyset$, there is a tree in $view(S_d \cup L_A)$ whose leaves $\{C_1, \dots, C_t\}$ are all evolvable for $\mathcal{P}_B$ and none of those credentials' policies is in S_d . So if all those credentials are unprotected, then there exists a full disclosure tree, which means that f and DTS are not compatible, leading to a contradiction. $\square$

Proof of theorem 7.3: $\forall G, L, R$, let $f'(G, L, R) = \{m_1, \dots, m_k\}$. Since f' is a hybrid of f_1 and f_2 , $m_i \in f_1(G, L, R)$ or $m_i \in f_2(G, L, R)$, for all $1 \leq i \leq k$. Because both f_1 and f_2 are in the DTS family, there exists $m' \in DTS(G, L, R)$ such that $m' \subseteq m_i$. Thus, $f' \succeq DTS$. $\square$

Proof of theorem 7.4: Because TrustBuilder-Simple $\succeq$ TrustBuilder-Relevant, it suffices to show that $DTS \preceq$ TrustBuilder-Relevant.

Suppose $\mathcal{P}_A$ is the local party and $TrustBuilder-Relevant(G, L_A, R) = \{m\}$. If $m = \emptyset$, that means $\mathcal{P}_A$ has no relevant policies and unlocked credentials other than those in S_d . Then $view(S_d)$ must have no evolvable trees for $\mathcal{P}_A$. Otherwise, suppose $T \in view(S_d)$ has an evolvable leaf C for $\mathcal{P}_A$. Then, neither C nor C 's policy is in S_d and C is syntactically relevant to R . If C is unlocked by credentials in S_d , then $C \in m$. Otherwise, C 's policy is in m . Both cases contradict the assumption that $m_r = \emptyset$.

When $m \neq \emptyset$, if $DTS(G, L_A, R) = \{\emptyset\}$, then $\emptyset \subseteq m$. Otherwise, according to definition 7.1, we must have $view(S_d \cup L_A) \neq \emptyset$ and $view(S_d)$ contains at least one evolvable tree for $\mathcal{P}_A$. Suppose $m = \{d_1, \dots, d_t\}, 1 \leq t$. If $view(S_d)$ already has an evolvable tree for $\mathcal{P}_B$, then $\{d_1\}$ is a non-empty minimal set such that $view(S_d \cup \{d_1\})$ has an evolvable tree for $\mathcal{P}_B$. So $\{d_1\} \in DTS(G, L_A, R)$ and $\{d_1\} \subseteq m$. On the other hand, if $view(S_d)$ has no evolvable tree for $\mathcal{P}_B$, consider any $m' \in DTS(G, L_A, R)$. Let d be any disclosure in m' . If d is the disclosure of a credential C , then C must appear in a disclosure tree in $view(S_d \cup (m' - \{d\}))$. Otherwise, $view(S_d \cup (m' - \{d\})) = view(S_d \cup m')$ which contradicts the fact that m' is a minimal set. By similar arguments, when d is a disclosure of credential C 's policy, we also have C appearing in a disclosure tree in $view(S_d \cup (m' - \{d\}))$. By proposition 7.1, C is syntactically relevant to R . Therefore, $d \in m$, which means $m' \subseteq m$. Thus, $DTS \preceq \text{TrustBuilder-Relevant}$. $\square$

Proof of theorem 7.5: For TrustBuilder-Simple, every time there are new credential disclosures in an incoming message, TrustBuilder-Simple needs to scan each resource's policy and check whether it is unlocked. Since there are at most n credential disclosures, such cost is bounded by $O(nm)$. To determine when to disclose a denial policy, TrustBuilder-Simple only needs to scan each incoming policy disclosure once, at a cost of no more than $O(m)$. Therefore, the total computation cost of TrustBuilder-Simple during trust negotiation is bounded by $O(nm)$.

Compared to TrustBuilder-Simple, besides the cost for checking whether resources are unlocked (with cost of $O(nm)$) and when to disclose a denial policy (with cost of $O(m)$), the only extra cost of TrustBuilder-Relevant is to determine whether a credential is relevant to R . At the beginning of the negotiation, R is the only known resource that is relevant to R . During the negotiation, every time there is a policy disclosure of a resource relevant to R , TrustBuilder-Relevant can scan the policy and mark all the credentials appearing in it as relevant. Similarly, every time TrustBuilder discloses a policy of a relevant resource, it can also mark those credentials appearing in the policy as relevant. By this way, in the whole negotiation process, all the policies are scanned only once to determine relevant resources. Therefore the cost is $O(m)$. Thus, the total computation cost of TrustBuilder-Relevant is also bounded by $O(nm)$. $\square$

B An Example of Trust Negotiation

Consider an online medical record service. The service allows patients to access their electronic medical records from Busey Hospital over the web. Now suppose a child's parent wants to access her child's medical records at the online service. The policies of the parent and the online service are shown in figure 8. Table 1 gives the interpretations of the credentials appearing in figure 8.

This example fits most naturally into a simple first-order language. In fitting it into a propositional format, we have trimmed off much of its natural complexity. In particular, several of the individual credentials (S_2, S_3, C_2, C_3, C_5 and C_6) would naturally be represented by credential chains, where the issuer of one credential is the owner of the next credential in the chain. For example, birth certificates are issued by county clerks, who are certified by their countries to their county clerks. In turn, the counties are certified by their states, possessions, or territories. The states/possessions/territories are certified by the Federal government, which is at the top of this particular set of credential chains.

Further, authentication of the requesters to one of the principals mentioned in the credentials is a key aspect of many of the policies here, but is not included in this propositional format. For example, in the policy $R \leftarrow (C_1 \wedge C_2) \vee (C_7 \wedge (C_3 \vee C_5 \vee C_6)) \vee C_4$, the requester must authenticate to the owner of C_1 and C_2 , and the owner of C_1 must be the patient whose records are being

requested, if the first clause is to be satisfied. To satisfy the second clause, the requester must authenticate to one of the parents in the child's birth certificate, or the guardian in the guardian credential, or one of the parents in the adoption credential. Further, the child named in the adoption/guardianship/birth certificate must be the owner of the child patient ID, and must be the patient whose records have been requested. To satisfy the third clause, the requester must authenticate to the owner of the employee ID.

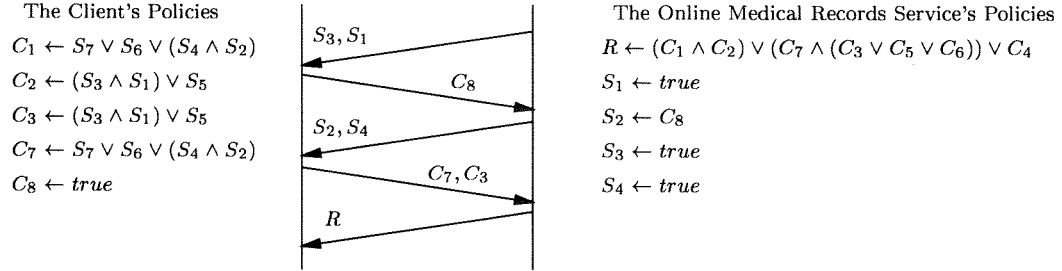


Figure 8: An example of access control policies and a safe disclosure sequence.

<i>Credential</i>	<i>Interpretation</i>
R	patient's medical record from Busey Hospital, stored at the Online Medical Record Service
S_1	TrustE membership credential
S_2	mailing address certification issued by the US Post Office, showing an address in Illinois
S_3	accreditation credential issued by the US Government to medical facilities
S_4	affiliate organization credential issued by Busey Hospital
S_5	US Government agency credential issued by the US Government
S_6	employee credential issued by Blue Cross Blue Shield of Illinois
S_7	employee ID issued by Busey Hospital
C_1	adult patient ID credential issued by Busey Hospital
C_2	adult's birth certificate (age over 21)
C_3	child's birth certificate (age under 21)
C_4	employee ID issued by Busey Hospital to a medical practitioner
C_5	legal guardianship credential issued by a state court
C_6	adoption credential issued by a state court
C_7	child patient ID credential issued by Busey Hospital
C_8	Reduce Junk Mail Alliance membership credential

Table 1: Interpretations of credentials appearing in figure 8.

Interoperable Strategies in Automated Trust Negotiation

Abstract

Automated trust negotiation is an approach to establishing trust between strangers through the exchange of digital credentials and the use of access control policies that specify what combinations of credentials a stranger must disclose in order to gain access to each local service or credential. We introduce the concept of a trust negotiation *protocol*, which defines the ordering of messages and the type of information messages will contain. To carry out trust negotiation, a party pairs its negotiation protocol with a trust negotiation *strategy* that controls the exact content of the messages, i.e., which credentials to disclose, when to disclose them, and when to terminate a negotiation. There are a huge number of possible strategies for negotiating trust, each with different properties with respect to speed of negotiations and caution in giving out credentials and policies. In the autonomous world of the Internet, entities will want the freedom to choose negotiation strategies that meet their own goals, which means that two strangers who negotiate trust will often not use the same strategy. To date, only a tiny fraction of the space of possible negotiation strategies has been explored, and no two of the strategies proposed so far will interoperate. In this paper, we define a large set of strategies called the *disclosure tree strategy (DTS) family*. Then we prove that if two parties each choose strategies from the DTS family, then they will be able to negotiate trust as well as if they were both using the same strategy. Further, they can change strategies at any point during negotiation. We also show that the DTS family is closed, i.e., any strategy that can interoperate with every strategy in the DTS family must also be a member of the DTS family. We also give examples of practical strategies that belong to the DTS family and fit within the TrustBuilder architecture and protocol for trust negotiation.

1 Introduction

With billions of users on the Internet, most interactions will occur between strangers, i.e., entities that have no pre-existing relationship and may not share a common security domain. In order for strangers to conduct secure transactions, a sufficient level of mutual trust must be established. For this purpose, the *identity* of the participants (e.g., their social security number, fingerprint, institutional tax ID) will often be irrelevant to determining whether or not they should be trusted. Instead, the *properties* of the participants, e.g., employment status, citizenship, group membership, will be most relevant. Traditional security approaches based on identity require a new client to pre-register with the service, in order to obtain a local login, capability, or credential before requesting service; but the same problem arises when the client needs to prove on-line that she is eligible to register with the service. E-commerce needs a more scalable approach that allows automatic on-line pre-registration, or does away entirely with the need for pre-registration. We believe that automated trust establishment is such a solution.

With automated trust establishment, strangers establish trust by exchanging *digital credentials*, the on-line analogues of paper credentials that people carry in their wallets: digitally signed assertions by a credential issuer about the credential owner. A credential is signed using the issuer's

private key and can be verified using the issuer's public key. A credential describes one or more attributes of the owner, using attribute name/value pairs to describe properties of the owner asserted by the issuer. Each credential also contains the public key of the credential owner. The owner can use the corresponding private key to answer challenges or otherwise demonstrate ownership of the credential. Digital credentials can be implemented using, for example, X.509 [10] certificates.

While some resources are freely accessible to all, many require protection from unauthorized access. Access control policies can be used for a wide variety of "protected" resources, such as services accessed through URLs, roles in role-based access control systems, and capabilities in capability-based systems. Since digital credentials themselves can contain sensitive information, their disclosure will often also be governed by access control policies. For example, suppose that a landscape designer wishes to order plants from Champaign Prairie Nursery (CPN). She fills out an order form on the web, checking an order form box to indicate that she wishes to be exempt from sales tax. Upon receipt of the order, CPN will want to see a valid credit card or her account credential issued by CPN, and a current reseller's license. The designer has no account with CPN, but she does have a digital credit card. She is willing to show her reseller's license to anyone, but she will only show her credit card to members of the Better Business Bureau. Therefore, when protected credentials are involved, a more complex procedure needs to be adopted to establish trust through negotiation.

2 Related Work

Credential-based authentication and authorization systems fall into three groups: identity-based, property-based, and capability-based. Originally, public key certificates, such as X.509 [10] and PGP [17], simply bound keys to names, and X.509 v.3 certificates later extended this binding to general properties (attributes). Such certificates form the foundation of identity-based systems, which authenticate an entity's identity or name and use it as the basis for authorization. Identity is not a useful basis for our aim of establishing trust among strangers.

Systems have emerged that use property-based credentials to manage trust in decentralized, distributed systems [8, 12, 15]. Johnson et al. [12] use attribute certificates (property-based credentials) and use-condition certificates (policy assertions) for access control. Use-condition certificates enable multiple, distributed stakeholders to share control over access to resources. In their architecture, the policy evaluation engine retrieves the certificates associated with a user to determine if the use conditions are met. Their work could use our approach to protect sensitive certificates.

The Trust Establishment Project at the IBM Haifa Research Laboratory [8] has developed a system for establishing trust between strangers according to policies that specify constraints on the contents of public-key certificates. Servers can use a collector to gather supporting credentials from issuer sites. Each credential contains a reference to the site associated with the issuer. That site serves as the starting point for a collector-controlled search for relevant supporting credentials. Security agents in our work could adopt the collector feature, and we could use their policy definition language. Their work could use our approach to protect sensitive credentials and gradually establish trust.

Capability-based systems manage delegation of authority for a particular application. Capability-based systems are not designed for establishing trust between strangers, since clients are assumed to possess credentials that represent authorization of specific actions with the application server. In the capability-based KeyNote system of Blaze et al. [2, 3], a credential describes the conditions under which one principal authorizes actions requested by other principals. KeyNote policies delegate authority on behalf of the associated application to otherwise untrusted parties. KeyNote

credentials express delegation in terms of actions that are relevant to a given application. KeyNote policies do not interpret the meaning of credentials for the application. This is unlike policies designed for use with property-based credentials, which typically derive roles from credential attributes. The IETF Simple Public Key Infrastructure [9] uses a similar approach to that of KeyNote by embedding authorization directly in certificates.

Bonatti et al. [4] introduced a uniform framework and model to regulate service access and information release over the Internet. Their framework is composed of a language with formal semantics and a policy filtering mechanism. Our work can be integrated with their framework.

The P3P standard [14] focuses on negotiating the disclosure of a user's sensitive private information based on the privacy practices of the server. Trust negotiation is generalized to base disclosure on any server property of interest to the client that can be represented in a credential. The work on trust negotiation focuses on certified properties of the credential holder while P3P is based on data submitted by the client that are claims the client makes about itself. Support for both kinds of information in trust negotiation is warranted.

SSL [7], the predominant credential-exchange mechanism in use on the web, and its successor TLS [5, 6] support credentials exchange during client and server authentication. The protocol is suited for identity-based credentials and would need extension to make it adaptable to property-based credentials. Needed additions include protection for sensitive server credentials and a way for the client to explain its policies to the server.

Islam et al. [11] show how to control downloaded executable content using policy graphs. Their system assumes that all the appropriate credentials accompany requests for downloaded content. Their work could be extended using our approach to disclose policies and conduct negotiations.

The first trust negotiation strategies proposed included a naive strategy that discloses credentials as soon as they are unlocked and discloses no policy information, as well as a strategy that discloses credentials only after each party determines that trust can be established, based on reviewing the other party's policies [15]. Yu et al. [16] introduced a new strategy that would succeed whenever success was possible and had certain efficiency guarantees. In [13], consideration was given for sensitive policy information in several strategies that established trust gradually through the introduction of policy graphs. The fact that none of the strategies proposed in this earlier work will interoperate demonstrates the need for trust negotiation protocols and strategy families to support interoperability between negotiation strategies.

3 Trust Negotiation

In our approach to automated trust establishment, trust is established incrementally by exchanging credentials and requests for credentials, an iterative process known as *trust negotiation*. While a *trust negotiation protocol* defines the ordering of messages and the type of information messages will contain, a *trust negotiation strategy* controls the exact content of the messages, i.e., which credentials to disclose, when to disclose them, and when to terminate a negotiation. Figure 1 introduces our *TrustBuilder* architecture for trust negotiation. Each participant in the negotiation has an associated security agent (SA) that manages the negotiation. The security agent mediates access to local protected *resources*, i.e., services and credentials. We say a credential or access control policy is *disclosed* if it has been sent to the other party in the negotiation, and that a service is disclosed if the other party is given access to it. Disclosure of protected resources is governed by access control policies. During a negotiation, the security agent uses a local negotiation strategy to determine what local resources to disclose next, and to accept new disclosures from the other party.

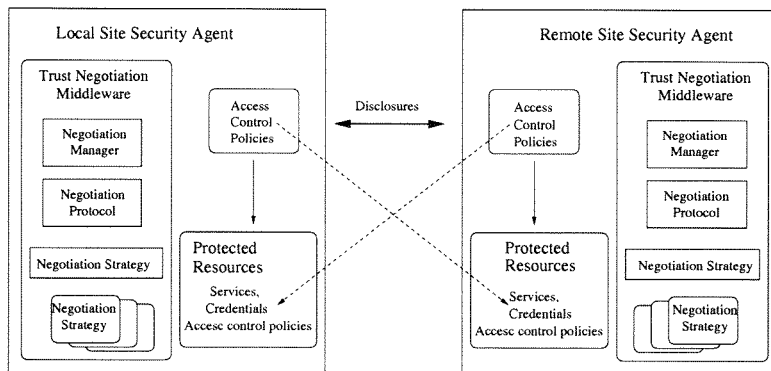


Figure 1: An architecture for automated trust negotiation. A security agent that manages local protected resources and their associated access control policies represents each negotiation participant. A access control policy specifies what resources the other party needs to disclose in order to gain access to a local resource, as indicated by the dotted lines in the figure. Trust negotiation middleware enables negotiation strategy interoperability.

The architecture in figure 1 supports a single protocol for establishing trust, and assumes there will be a variety of negotiation strategies that must be supported. All trust negotiation strategies share the goal of building trust through an exchange of digital credentials that leads to obtaining access to a protected resource. Once enough trust has been established that a particular credential can be disclosed to the other party, a local negotiation strategy must determine whether the credential is relevant to the current stage of the negotiation. Different negotiation strategies will use different definitions of relevance, involving tradeoffs between computational cost, the length of the negotiation, and the number of disclosures.

From the handful of trust negotiation strategies proposed so far in the literature, it is clear that there are endless possible variations in how to negotiate trust. Rather than exploring the space of all possible strategies one strategy at a time, our goal in this paper is to characterize a broad class of strategies (section 6) and design a strategy-independent, language-independent trust negotiation protocol (section 5) that ensures their interoperability within the TrustBuilder trust negotiation architecture.

4 Access Control Policies

We assume that the information contained in access control policies (*policies*, for short) and credentials can be expressed as finite sets of statements in a formal language with a well-defined semantics. XML or logic programming languages with appropriate semantics may be suitable languages in practice [8, 1]. For convenience, we will assume that the original language allows us to describe the meaning of a set of statements as the set of all models that satisfy the set of statements, in the usual logic sense. We say that a set X of statements *satisfies* a set of statements P if and only if P is true in all models of X . For purely practical reasons, we require that the language be *monotonic*, i.e., if a set of statements X satisfies policy P , then any superset of X will also satisfy P ; that way, once a negotiation strategy has determined that the credentials disclosed by a participant satisfy the policy of a resource, the strategy knows that the same policy will be satisfied for the rest of the negotiation, and does not have to be rechecked.

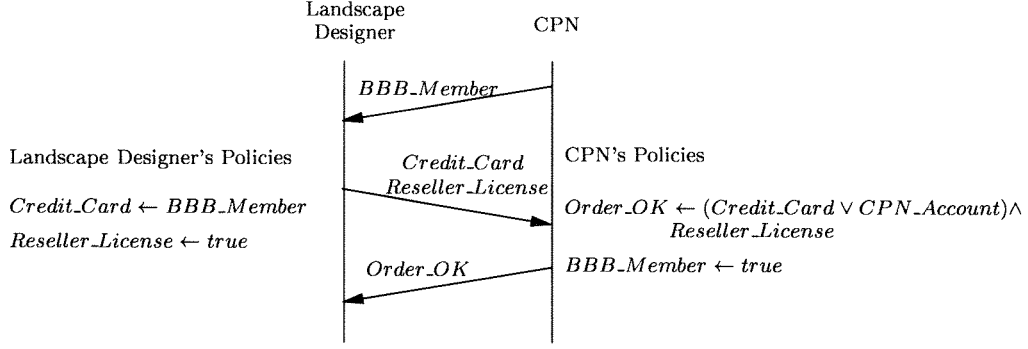


Figure 2: An example of access control policies and a safe disclosure sequence which establishes trust between the server and the client.

In this paper, we will treat credentials and services as propositional symbols. Each of these resources has exactly one access control policy, of the form $C \leftarrow F_C(C_1, \dots, C_k)$, where $F_C(C_1, \dots, C_k)$ is a Boolean expression involving only credentials $C_1, \dots, C_k$ that the other party may possess, Boolean constants *true* and *false*, the Boolean operators $\vee$ and $\wedge$, and parentheses as needed. C_i is satisfied if and only if the other party has disclosed credential C_i . We assume that we can distinguish between local and remote resources (by renaming propositional symbols as necessary). Resource C is *unlocked* if its access control policy is satisfied by the set of credentials disclosed by the other party. A resource is *unprotected* if its policy is always satisfied. The *denial policy* $C \leftarrow false$ means that either the party does not possess C , or else will not disclose C under any circumstances. A party implicitly has a denial policy for each credential it does not possess. If the disclosure of a set S of credentials satisfies resource R 's policy, then we say S is a *solution set* for R . Further, if none of S 's proper subsets is a solution set for R , we say S is a *minimal solution set* for R . The *size* of a policy is the number of symbol occurrences in it.

Given sequence $G = (C_1, \dots, C_n)$ of disclosures of protected resources, if each C_i is unlocked at the time it is disclosed, $1 \leq i \leq n$, then we say G is a *safe disclosure sequence*. The goal of trust negotiation is to find a safe disclosure sequence $G = (C_1, \dots, C_n = R)$, where R is the resource to which access was originally requested. When this happens, we say that trust negotiation succeeds. If $C_i = C_j$ and $1 \leq i < j \leq n$, then we say G is *redundant*. Since the language used to represent policies and credentials is monotonic, we can remove the later duplicates from a redundant safe disclosure sequence and the resulting sequence is still safe. Figure 2 shows a safe disclosure sequence for the landscape designer's purchase from CPN discussed earlier. A more complex example can be found in Appendix B. It is important to note that this example, and our algorithms that follow, rely on *lower levels of software* to perform the functions associated with disclosure of a credential: verification of its contents, checks for revocation as desired, checks of validity dates, authentication of ownership, etc., as is normally done for X.509 certificates. The necessary calls to these functions, and the translation of credentials into the language used to represent policies, are not shown in our algorithms.

5 The TrustBuilder Protocol and Strategy Families

Previous work on trust negotiation has not explicitly proposed any trust negotiation protocols, instead defining protocols implicitly by the way each negotiation strategy works. This is one reason

```

TrustBuilder.handle_disclosure_message ( $m, R$ )
Input:  $m$  is the last disclosure message received from the remote party.
       $R$  is the resource to which the client originally requested access.
TrustBuilder.check_for_termination( $m, R$ ). //Stop negotiating, if appropriate.
TrustBuilder.next_message( $m, R$ ).
End of TrustBuilder.handle_disclosure_message.

TrustBuilder.next_message( $m, R$ )
// First, let the local strategy suggest what the next message should be.
Let  $G$  be the disclosure message sequence so far.
Let  $L$  be the local resources and policies.
 $S_m = \text{Local\_strategy}(G, L, R)$ .
//  $S_m$  contains the candidate messages the local strategy suggests.
Choose any single message  $m'$  from  $S_m$ .
Send  $m'$  to the remote party.
TrustBuilder.check_for_termination( $m', R$ ). //Stop negotiating, if appropriate.
End of TrustBuilder.next_message.

TrustBuilder.check_for_termination( $m, R$ )
If  $m$  is the empty set  $\emptyset$  and this is not the beginning of the negotiation,
    Then negotiations have failed. Stop negotiating and exit.
If  $m$  contains the disclosure of  $R$ ,
    Then negotiations have succeeded. Stop negotiating and exit.
End of TrustBuilder.check_for_termination.

```

Figure 3: Pseudocode for the TrustBuilder protocol. The negotiation is triggered when the client asks to access a protected resource owned by the server. After rounds of disclosures, either one party sends a failure message and ends the negotiation, or the server grants the client access.

why no two different previously proposed strategies can interoperate – their underlying protocols are totally different.

We remedy this problem by defining a simple protocol for TrustBuilder. Formally, a *message* in the TrustBuilder protocol is a set $\{R_1, \dots, R_k\}$ where each R_i is a disclosure of a local credential, a local policy, or a local resource. When a message is the empty set $\emptyset$, we also call it a *failure message*. Further, to guarantee the safety and timely termination of trust negotiation no matter what policies and credentials the parties possess, the TrustBuilder protocol requires the negotiation strategies used with it to enforce the following three conditions throughout negotiations:

1. If a message contains a denial policy disclosure $C \leftarrow \text{false}$, then C must appear in a previously disclosed policy.
2. A credential or policy can be disclosed at most once.
3. Every disclosure must be safe.

Before the negotiation starts, the client sends the original resource request message to the server indicating its request to access resource R . This request triggers the negotiation, and the server invokes its local security agent with the call `TrustBuilder.handle_disclosure_message( $\emptyset, R$ )`. Then the client and server exchange messages until either the service R is disclosed by the server or one party sends a failure message. The whole negotiation process is shown in figure 3.

In the remainder of this paper, unless otherwise noted, we discuss only strategies that can be called from the TrustBuilder protocol and satisfy the three conditions above. A formal definition of a negotiation strategy is given below.

Definition 5.1. A strategy is a function $f : \{G, L, R\} \rightarrow S_m$, where R is the resource to which the client originally requested access, $G = (m_1, \dots, m_k)$ is a sequence of disclosure messages such that $m_i \neq \emptyset$ and $R \notin m_i$ for $1 \leq i \leq k$, L is the set of local resources and policies, and S_m is a set of disclosure messages. Further, every disclosure in a message in S_m must be of a local resource or policy, as must be all the disclosures in m_{k-2i} , for $1 \leq k-2i < k$. The remaining disclosures in G are of remote resources and policies.

Intuitively, based on the disclosures made so far, plus the local resources and policies, a negotiation strategy will suggest the next set of disclosures to send to the other party. Note that a strategy returns a *set* of possible disclosure messages, rather than a single message. Practical negotiation strategies will suggest a single next message, but the ability to suggest several possible next messages will be very convenient in our formal analysis of strategy properties, so we include it both in the formal definition of a negotiation strategies and also in the protocol pseudocode in figure 3.

Definition 5.2. Strategies f_A and f_B are compatible if whenever there exists a safe disclosure sequence for a party $\mathcal{P}_A$ to obtain access to a resource owned by party $\mathcal{P}_B$, the trust negotiation will succeed when $\mathcal{P}_A$ uses f_A and $\mathcal{P}_B$ uses f_B . If $f_A = f_B$, then we say that f_A is self-compatible.

Definition 5.3. A strategy family is a set $\mathcal{F}$ of mutually compatible strategies, i.e., $\forall f_1 \in \mathcal{F}, f_2 \in \mathcal{F}$, f_1 and f_2 are compatible. We say a set $\mathcal{F}$ of strategies is closed if given a strategy f' , if f' is compatible with every strategy in $\mathcal{F}$, then $f' \in \mathcal{F}$.

One obvious advantage of strategy families is that a security agent (SA) can choose strategies based on its needs without worrying about interoperability, as long as it negotiates with other SAs that use strategies from the same family. As another advantage, under certain conditions, an SA does not need to stick to a fixed strategy during the entire negotiation process. It can adopt different strategies from the family in different phases of the negotiation. For example, during the early phase, since the trust between two parties is very limited, an SA may adopt a cautious strategy for disclosing credentials. When a certain level of trust has been established, in order to accelerate the negotiation, the SA may adopt a less cautious strategy. However, without the closure property, a family may not be large enough for practical use. As an extreme example, given any self-compatible strategy f , $\{f\}$ is a strategy family. The closure property guarantees the maximality of a strategy family.

In general, the notions of strategy families and closed sets of strategies are incomparable, in the sense that neither of them implies the other. For example, if a strategy f 's output is $\{m\}$, where m is a message containing all the undisclosed local policies and unlocked credentials, then it is easy to prove that f is self-compatible. Then $\{f\}$ is a family, but by no means closed. On the other hand, consider the strategy f' whose output is always $\{\emptyset\}$. Obviously f' is not compatible with any strategies. The strategy set $\{f'\}$ is not a family but is closed.

We end this section with two simple propositions.

Proposition 5.1. Any subset of a strategy family is also a family.

Proposition 5.2. If a strategy family $\mathcal{F}$ is a proper subset of another family, then $\mathcal{F}$ is not closed.

6 Characterizing Safe Disclosure Sequences

In this section, we define the concepts that we use to describe the progress of a negotiation and to characterize the behavior of different strategies. In the remainder of the paper, we use R to represent the resource to which access was originally requested.

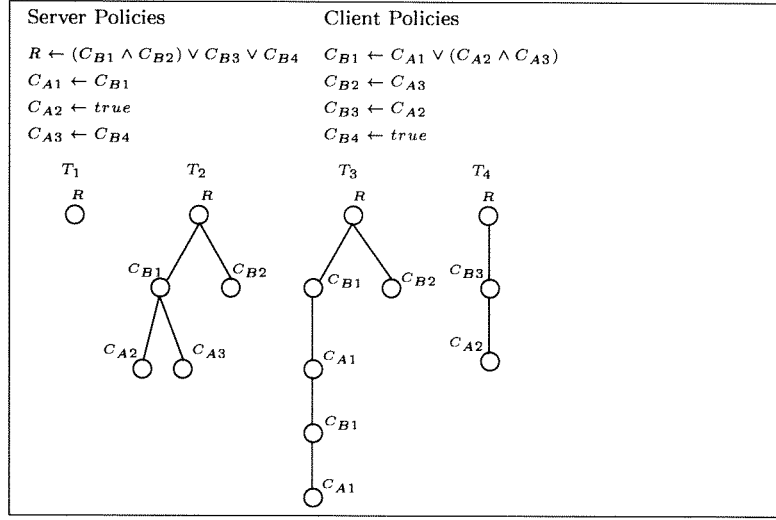


Figure 4: Example disclosure trees for a set of policies

6.1 Disclosure Trees

Definition 6.1. A disclosure tree for R is a finite tree satisfying the following conditions:

1. The root represents R .
2. Except for the root, each node represents a credential. When the context is clear, we refer to a node by the name of the credential it represents.
3. The children of a node C form a minimal solution set for C .

When all the leaves of a disclosure tree T are unprotected credentials, we say T is a full disclosure tree. Given a disclosure tree T , if there is a credential appearing twice in the path from a leaf node to the root, then we call T a redundant disclosure tree.

Figure 4 shows example disclosure trees. Note that T_2 is redundant and T_4 is a full disclosure tree.

The following theorems state the relationship between disclosure trees and safe disclosure sequences that lead to the granting of access to resource R . Proofs of all theorems can be found in Appendix A.

Theorem 6.1. Given a non-redundant safe disclosure sequence $G = (C_1, \dots, C_n = R)$, there is a full non-redundant disclosure tree T such that both of the following hold:

1. The nodes of T are a subset of $\{C_1, \dots, C_n\}$.
2. For all credential pairs (C'_1, C'_2) such that C'_1 is an ancestor of C'_2 in T , C'_2 is disclosed before C'_1 in G . □

Theorem 6.2. Given a full disclosure tree for R , there is a non-redundant safe disclosure sequence ending with the disclosure of R . □

By theorems 6.1 and 6.2, we get the following corollary immediately.

Corollary 6.1. *Given a safe disclosure sequence $G = (C_1, \dots, C_n = R)$, there is a full non-redundant disclosure tree T such that:*

1. *The credential nodes of T are a subset of $\{C_1, \dots, C_n\}$,*
2. *For all credential pairs (C'_1, C'_2) such that C'_1 is an ancestor of C'_2 in T , the first disclosure of C'_2 in G is before the first disclosure of C'_1 .*

Without loss of generality, from now on, we consider only non-redundant disclosure sequences.

Since there is a natural mapping between safe disclosure sequences and disclosure trees, during the negotiation, theoretically one could determine whether a potential credential or policy disclosure is helpful by examining all the disclosure trees for R . At the beginning of a negotiation, before disclosures begin, the only relevant disclosure tree for the client contains a single node R . As the negotiation proceeds, other trees may become relevant. The following definitions help us describe the set of relevant trees.

Definition 6.2. *Given a disclosure tree T and a set of S_c of credentials, the reduction of T by S_c , $\text{reduction}(T, S_c)$, is the disclosure tree T' which is obtained by removing all the subtrees rooted at a node representing resource $C \in S_c$. Given a set S_t of disclosure trees, $\text{reduction}(S_t, S_c) = \{\text{reduction}(T, S_c) \mid T \in S_t\}$.*

If S_c is the set of credential disclosures made so far, then reducing T by S_c prunes out the part of the negotiation that has already succeeded. Intuitively, if a credential C has been disclosed, then we already have a safe disclosure sequence for C . We do not need to disclose additional credentials or policies in order to get a full disclosure tree rooted at C . An example of a disclosure tree reduction is shown in figure 5(a).

Definition 6.3. *Given a disclosure tree T and a policy set S_p containing no denial policies, the expansion of T by S_p , $\text{expansion}(T, S_p)$, is the set of all disclosure trees T_i such that*

1. *T is a subgraph of T_i , i.e., there exists a set S of credentials such that $\text{reduction}(T_i, S) = T$.*
2. *For each edge (C_1, C_2) in T_i , if (C_1, C_2) is not an edge of T , then C_1 's policy is in S_p .*
3. *For each leaf node C of T_i , either S_p does not contain C 's policy, or T_i is redundant.*

Given a set of disclosure trees S_t , $\text{expansion}(S_t, S_p) = \bigcup_{T \in S_t} \text{expansion}(T, S_p)$.

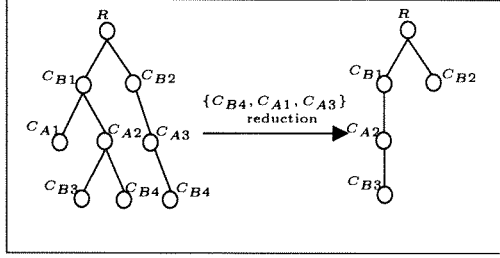
A disclosure tree can expand when a party receives new policy disclosures. An example of a disclosure tree expansion is shown in figure 5(b).

Definition 6.4. *Given a set S_t of disclosure trees and a set S_{dp} of denial policies, the denial pruning of S_t by S_{dp} , denoted $\text{pruned}_{\text{denial}}(S_t, S_{dp})$, is the set*

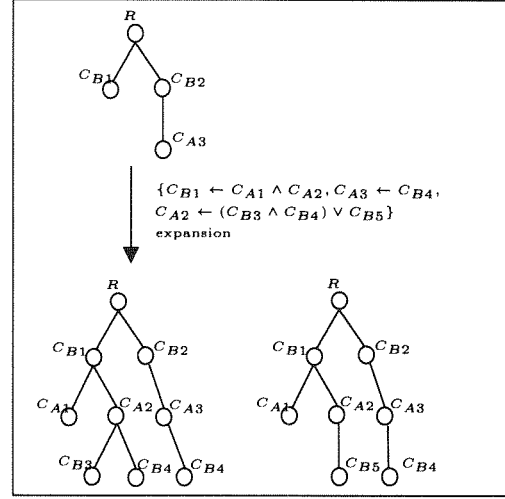
$$\{T \mid T \in S_t \text{ and } T \text{ contains no resource whose policy is in } S_{dp}\}.$$

Since a full disclosure tree contains only credentials that the two parties possess, if a disclosure tree node represents a credential with a denial policy, that tree cannot evolve into a full disclosure tree, and is no longer relevant.

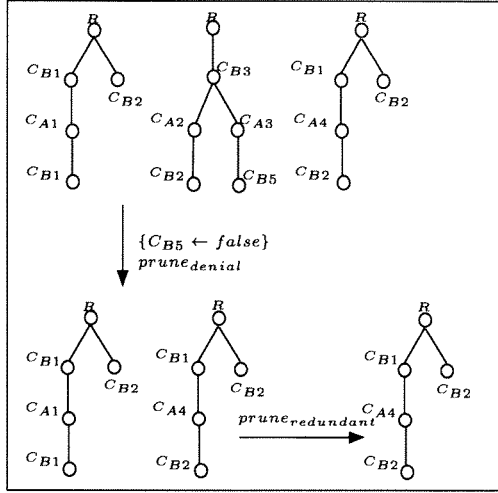
Definition 6.5. *Given a set S_t of disclosure trees, the redundancy pruning of S_t , denoted $\text{prune}_{\text{redundant}}(S_t)$, is the set*



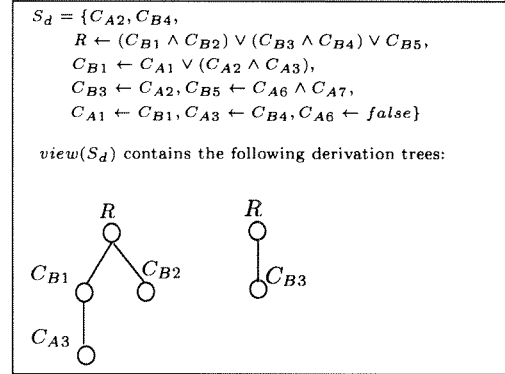
(a) Example of a disclosure tree reduction



(b) Example of a disclosure tree expansion



(c) Example of denial pruning and redundancy pruning



(d) Example of a view

Figure 5: Examples of operations on disclosure trees

$$\{T \mid T \in S_t \text{ and } T \text{ is not a redundant disclosure tree}\}.$$

The rationale for redundancy pruning will be shown after we introduce more operations on disclosure trees. Examples of denial and redundancy pruning are shown in figure 5(c).

Definition 6.6. Given a disclosure tree T and a set S_{dp} of denial policies, S_p of non-denial policies, and S_c of credentials, let $S = S_{dp} \cup S_p \cup S_c$. The evolution of T by S , denoted $\text{evolution}(T, S)$, is

$$\text{prune}_{\text{redundant}}(\text{prune}_{\text{denial}}(\text{reduction}(\text{expansion}(T, S_p), S_c), S_{dp})).$$

Given a set S_t of disclosure trees, $\text{evolution}(S_t, S) = \bigcup_{T \in S_t} \text{evolution}(T, S)$. As a special case, when T is the disclosure tree containing only a root node R , then we say $\text{evolution}(T, S)$ is the view of S , denoted $\text{view}(S)$.

During the negotiation, let S be the set of credentials and policies disclosed so far and L be the local policies of a negotiation party. Then $\text{view}(S \cup L)$ contains all the relevant disclosure trees which can be seen by this party. An example view is shown in figure 5(d). Sometimes even though a tree may evolve into a full tree later in the negotiation, it is nonetheless redundant and can be removed by redundancy pruning, whose correctness is guaranteed by the following theorem.

Theorem 6.3. Let T be a full but redundant disclosure tree. Then there is a full disclosure tree T' that is not redundant. $\square$

Suppose S is the set of currently disclosed credentials and policies. By theorem 6.3, if a redundant tree may evolve into a full tree, then the corresponding non-redundant tree is already included in $\text{view}(S)$. So the redundant trees are not relevant for the remainder of the negotiation.

In order to make a negotiation successful whenever the policies of the two negotiation parties allow, the negotiation strategy should make sure no possible full disclosure trees have been overlooked. A disclosure tree also tells a party what may contribute to the success of a negotiation. As an example, suppose party $\mathcal{P}_B$ requests service R from party $\mathcal{P}_A$. S_d , the set of disclosures so far, and $\text{view}(S_d)$ are shown in figure 6. Suppose now it is $\mathcal{P}_A$'s turn to send a message to $\mathcal{P}_B$. From the disclosure tree, it is clear to an outside observer that credentials C_{A1} and C_{A2} must be disclosed if the negotiation is to succeed. So $\mathcal{P}_A$'s negotiation strategy can now disclose C_{A1} 's and/or C_{A2} 's policy. This example shows that in order to let a negotiation party $\mathcal{P}$ know what might be the next appropriate message, a disclosure tree should have at least one leaf node that is a credential that the other party wants $\mathcal{P}$ to disclose. We have the following definition:

Definition 6.7. Disclosure tree T 's evolvable leaves for party $\mathcal{P}_A$, denoted $\text{evolvable}(T, \mathcal{P}_A)$, are the set of leaf nodes C of T such that either $C = R$ and $\mathcal{P}_A$ is the server, or C appears in a policy that $\mathcal{P}_B$ disclosed to $\mathcal{P}_A$. If $\text{evolvable}(T, \mathcal{P}_A) \neq \emptyset$, we say T is evolvable for $\mathcal{P}_A$.

The disclosure tree in figure 6 is evolvable for both $\mathcal{P}_A$ and $\mathcal{P}_B$.

If a negotiation reaches a point where every leaf node of some disclosure tree is unlocked, then the tree is a full tree and corresponds to a safe disclosure sequence.

Definition 6.8. Let $\mathcal{P}_A$ be a negotiation party, T be a disclosure tree, and S be a set of policies and credentials. If every resource in $\text{evolvable}(T, \mathcal{P}_A)$ is unlocked by credentials in S , then we say T is semi-full with S for $\mathcal{P}_A$. Further, we say T is full with S iff every leaf node of T is unlocked by credentials in S .

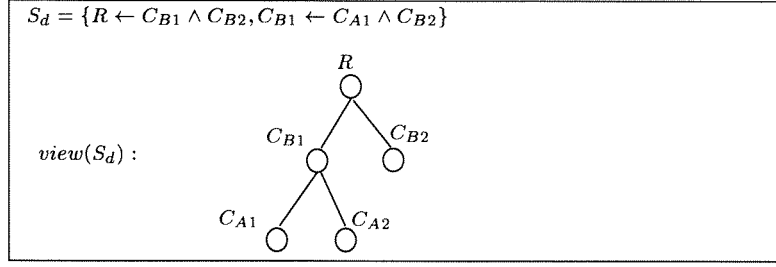


Figure 6: $view(S_d)$ where $S_d = \{R \leftarrow C_{B1} \wedge C_{B2}, C_{B1} \leftarrow C_{A1} \wedge C_{A2}\}$

6.2 Strategy Caution and Strategy Set Generators

If $\mathcal{F}$ is a strategy family, then intuitively, every strategy in $\mathcal{F}$ always discloses enough information to keep the negotiation moving towards success, if success is possible. If $\mathcal{F}$ is also closed, then $\mathcal{F}$ must also contain those strategies that disclose only the minimal amount of information needed to continue negotiations. Therefore it is helpful to formally define a relationship between strategies based on the information they disclose.

Definition 6.9. *Given two negotiation strategies f_1 and f_2 , if for all possible inputs G , L , and R to f_1 and f_2 , we have*

$$\forall m \in f_2(G, L, R) \exists m' \in f_1(G, L, R) \text{ such that } m' \subseteq m$$

then we say f_1 is at least as cautious as f_2 , denoted as $f_1 \preceq f_2$ or $f_2 \succeq f_1$.

Caution defines a partial order between strategies. Intuitively, if $f_2 \succeq f_1$ then f_2 always discloses at least as much information as f_1 does.

Definition 6.10. *Given a strategy f , the set of strategies generated by f , denoted $StraSet(f)$, is the set $\mathcal{F} = \{f' | f' \succeq f\}$. f is called the generator of $\mathcal{F}$.*

As we discussed in section 6.1, during a trust negotiation, evolvable trees give guidance on what a party needs to disclose in the next message so that the whole negotiation advances towards potential success. If there is no evolvable tree, then a cautious party will choose to end the negotiation even if the policies of the two parties allow success. Therefore, to ensure that negotiations succeed whenever possible, a strategy must ensure that the other party will have an evolvable tree when the other party needs to make its next disclosure. The only exception is when the strategy knows that no disclosure tree can evolve into a full tree.

7 The Disclosure Tree Strategy Family

We present the *disclosure tree strategy* (DTS), then prove that DTS generates a closed family. Throughout this section, we assume that $G = (m_1, \dots, m_k)$ is a sequence of messages such that $m_i \neq \emptyset$ and $R \notin m_i$ for $1 \leq i \leq k$. We assume L_A and L_B are the local policies of parties $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively, and $S_d = \bigcup_{1 \leq i \leq k} m_i$. Without loss of generality, we assume $\mathcal{P}_A$ will send the next message to $\mathcal{P}_B$.

Definition 7.1. *The Disclosure Tree Strategy (DTS for short) is a strategy $DTS(G, L_A, R)$ such that:*

- 1) $DTS(G, L_A, R) = \{\emptyset\}$ if and only if $\text{view}(S_d \cup L_A) = \emptyset$ or $\text{view}(S_d)$ has no evolvable tree for $\mathcal{P}_A$.
- 2) Otherwise, $DTS(G, L_A, R)$ contains all messages m' such that one of the following conditions holds:
 - $m' = \{R\}$, if R is unlocked by credentials in S_d ;
 - m' is a non-empty set of credentials and policies such that $\text{view}(S_d \cup m')$ contains at least one evolvable tree for $\mathcal{P}_B$, and no non-empty proper subset of m' has this property.

Condition 1) states under what circumstances the DTS strategy will terminate the negotiation with a failure message. Condition 2) guarantees that the other party will have an evolvable tree. Therefore, the other party can always send a message back that evolves a disclosure tree. Thus, no failure message will be sent unless there is no disclosure tree at all, in which case the negotiation cannot succeed anyway. Formally, we have the following theorems:

Theorem 7.1. *The set of strategies generated by DTS is a family.* □

Theorem 7.2. *If a strategy f and DTS are compatible, then $f \succeq DTS$.* □

We call the family generated by DTS the *DTS family*. By theorems 7.1 and 7.2, we get the following corollary immediately.

Corollary 7.1. *The DTS family is closed.* □

As we mentioned in section 5, one advantage of a strategy family can be the ability to adopt different strategies from a family in different phases of the negotiation. Correct interoperability is guaranteed as long as both parties' strategies are from the same family.

Definition 7.2. *Let f_1 and f_2 be two strategies. A strategy f' is a hybrid of f_1 and f_2 if $\forall G, L, R$, $f'(G, L, R) \subseteq f_1(G, L, R) \cup f_2(G, L, R)$ and $f' \neq f_1$ and $f' \neq f_2$.*

If a security agent adopts different DTS family strategies in different phases of trust negotiation, it is equivalent to adopting a hybrid of those strategies.

Theorem 7.3. *Let f_1 and f_2 be strategies in the DTS family and let f' be a hybrid of f_1 and f_2 . Then f' is also in the DTS family.* □

Therefore, as long as both parties use strategies from the DTS family, they can switch between different practical strategies as often as they like, and trust negotiation will still succeed whenever possible.

Although disclosure trees are a useful tool for understanding strategy properties, it would require exponential time and space to materialize all the disclosure trees during a negotiation. Fortunately, many strategies in the DTS family are quite efficient. We present two efficient strategies: TrustBuilder-Simple and TrustBuilder-Relevant, which are both in the DTS family.

The TrustBuilder-Simple strategy puts all undisclosed policies and unlocked credentials in the next message to the other party. If all the policies and unlocked credentials have already been disclosed, it will send a failure message. Its pseudocode is shown in figure 7(a).

We say a credential C is *syntactically relevant* to resource R iff C appears in R 's policy, or C appears in the policy of a credential C' that is relevant to R . In contrast to TrustBuilder-Simple, the TrustBuilder-Relevant strategy (figure 7(b)) discloses a credential C 's policy only if C is syntactically relevant to R . Similarly, TrustBuilder-Relevant only discloses syntactically relevant unlocked credentials.

TrustBuilder-Simple strategy*Input:* $G = (m_1, \dots, m_k)$: a sequence of safe disclosure messages. L : the local resources and policies of this party. R : the resource to which access was originally requested.*Output:*A set containing a single disclosure message m .*Pre-condition:* R has not been disclosed and $m_k \neq \emptyset$.Let $\mathcal{P}_A$ be the local party and $\mathcal{P}_B$ the remote party. $S_d = \bigcup_{1 \leq i \leq k} m_i$; $m = \emptyset$;For every local credential C that is unlocked by S_d $m = m \cup \{C\}$;For every local locked credential C if (C 's policy P is not a denial policy)then $m = m \cup \{P\}$;For every policy $P' \in S_d$ such that $P' \notin L$ For every credential that C appears in P' and has a denial policy $m = m \cup \{C \leftarrow false\}$; $m = m - S_d$;return $\{m\}$;

(a) Pseudocode for the TrustBuilder-Simple strategy

TrustBuilder-Relevant-Strategy*Input:* $G = (m_1, \dots, m_k)$: a sequence of safe disclosure messages. L : the local resources and policies of this party. R : the resource to which access was originally requested.*Output:*A set containing a single disclosure message m .*Pre-condition:* R has not been disclosed and $m_k \neq \emptyset$.Let $\mathcal{P}_A$ be the local party and $\mathcal{P}_B$ the remote party. $S_d = \bigcup_{1 \leq i \leq k} m_i$; $m = \emptyset$;For every local credential C syntactically relevant to R if (C is unlocked by S_d)then $m = m \cup \{C\}$;else $m = m \cup \{C's\ policy\}$;For every policy $P' \in S_d$ such that $P' \notin L$ For every credential C that appears in P' and has a denial policy $m = m \cup \{C \leftarrow false\}$; $m = m - S_d$;return $\{m\}$;

(b) Pseudocode for the TrustBuilder-Relevant strategy

Figure 7: Pseudocode for two strategies in the DTS family

Proposition 7.1. *If a credential C appears in a disclosure tree for R , then C is relevant to R .*

Theorem 7.4. *TrustBuilder-Simple and TrustBuilder-Relevant belong to the DTS family.* $\square$

Theorem 7.5. *The computation costs of TrustBuilder-Simple and TrustBuilder-Relevant in the whole process of trust negotiation are bounded by $O(nm)$, where n is the total number of credentials and m is the total size of the policies of both parties.* $\square$

The worst-case behavior of TrustBuilder-Simple and TrustBuilder-Relevant occurs when every credential belonging to one party appears in every policy belonging to the other party, and each disclosure message discloses a single credential or policy.

8 Access Control Policy Graphs

In our discussion so far, we made the assumption that each resource C is protected by a single access control policy which can be shown to others who requires access to C . However, as shown in the following examples, in some situations, even an access control policy itself may contain sensitive information and needs to be protected from revealing to strangers.

Example 1. A corporate web server manages information for a collaborative project between the corporation and a secret business partner. The information is accessible only to members of the project team from both companies. To obtain authorized access, team members must submit employee credentials that show that they work in one of the departments associated with the project. Since the access control policy includes information about a business relationship and a secret project, disclosing that policy to a stranger is undesirable.

A solution to this problem is for the server to begin trust negotiation by requesting an employee credential, which the server checks to see if the client works for the corporation or for the business partner, without the server divulging sensitive information that might disclose the nature of the information the server manages.

Example 2. A corporate web server provides a protected service intended for vice-presidents in Company A and all employees in Company B. Revealing the vice-president constraint to strangers unnecessarily raises interest or concerns regarding the rationale behind the constraint, especially by those who fail to satisfy the constraint. This problem can be solved in the same way as example 1.

Example 3. A web server provides access to sensitive corporate information for the corporation's suppliers. The corporation issues a credential to each supplier organization. Suppliers in turn issue credentials locally to their employees. Suppliers are autonomous, and each supplier has its own approach to issuing credentials. For instance, some suppliers may issue employee credentials at the corporate level, while others will issue site credentials at the site level. Suppliers will formulate their own policies about which credentials can be used to authenticate their employees. This situation creates the need for supplier-specific access control policies. If the trust negotiation system supports a single access control policy that combines all the supplier-specific policies, outsiders could learn about the security requirements of all the trusted suppliers. In addition, the trusted suppliers would learn about each other's policies, which may not be desirable. The solution is for the server initially to ask for a credential chain that includes the supporting supplier credential. Once the server knows which supplier the client is associated with, the server can release the supplier-specific access control policy information.

Example 4. A web server for a multinational corporation automatically supports benefits enrollment and payroll processing services. The available credentials and associated policies vary dramatically from country to country. To ensure that its policies are scalable and maintainable, and to keep message size reasonably small, the server requests a credential indicating the client's country of residence. Then it provides the policy associated with that country only.

These scenarios show that the mere mention of sensitive credentials, and the constraints imposed upon them in an access control policy, can leak sensitive corporate information. Adversaries might make use of that information in attempts to gain illicit access. Even when a policy is disclosed to an entity that eventually proves to be trustworthy, disclosing all the constraints to it may be undesirable. Subdivision of policies also has advantages for performance and policy maintainability. Therefore, an access control policy should also be regarded as a resource and its disclosure needs to be protected by some other access control policies. A resource may be guarded by layers of access control policies instead of only a single policy. We introduce the concept of *access control policy graphs* (*policy graphs*, for short) to represent such layers of access control policies.

In the following of the paper, a protected resource can be a service, a policy, or a credential. A policy graph for a protected resource R is a finite directed acyclic graph with a single source node S and a single sink R . (As mentioned in section 4, we assume that the name of a node is the name of the resource it represents.) All the nodes except R represent policies that specify the properties that a negotiation participant may be required to demonstrate in order to gain access to R . Each sensitive credential and service will have its own separate policy graph. Each policy represented as a node in a policy graph G implicitly also has its own graph — the maximum subgraph of G for which that policy node is the sole sink. It is *safe* for a party to disclose a resource R only if there is a path from the source to R in R 's policy graph such that every policy node (except R when R represents a policy) in the path is *satisfied* by the disclosed credentials so far. It is easy to see that when a resource is protected only by a single policy, as we discussed in previous sections, its policy graph will only have two nodes: a sink node (which represents the resource) and a source node (which represents the policy). Example policy graphs for examples 1-4 are shown in figure 8.

Although policy graphs are motivated by sensitive policy information, they have other advantages in cases when a policy graph is extremely large, as in example 4. Only the relevant subset of the graph need be disclosed, potentially saving communications and storage resources. For example 4 in figure 8, a client will first be sent the policy asking for an IBM employee credential. If one is supplied, the client is then asked for a passport. (the policy does not compare the names on the two credentials, instead assuming that anyone who can pass the authentication challenges for both credentials is the individual referred to by both credentials.) The server then immediately determine which successor node is satisfied, based on the passport's issuing country, without further communication with the client. The process will then move on to the appropriate subtree. Example 3 in figure 8 is quite similar; here we see how the subtrees join together again. In example 1 in figure 3, the companies Microsoft and IBM are not mentioned in the source node, instead appearing only in its immediate successors. The employee credential supplied to satisfy the source node will allow the successor nodes to be evaluated without further communication with the client, who will not see the policies in the successor nodes. (If the company names should not be secret, then the source node should also require that the company name be IBM or Microsoft.) Example 2 in the same figure is very similar.

Note that the policies of those examples in figure 8 are represented in a language based on first-order logic without quantifiers or negation. To comply with the discussion in previous sections, in the remainder of the paper, we switch to propositional logic without negation. This simplicity

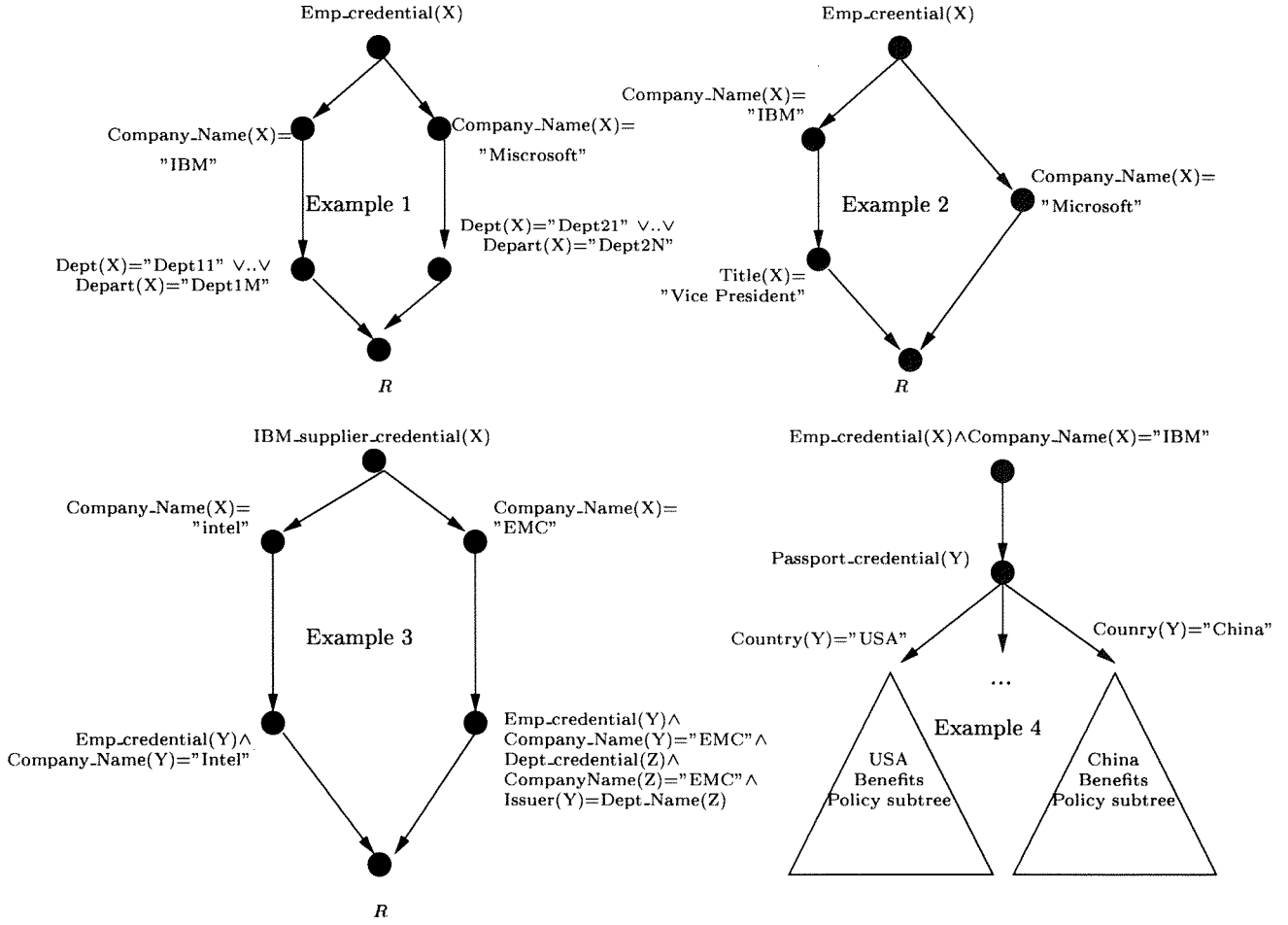


Figure 8: Policy graphs for examples 1-4, using a subset of first-order logic with a graph-based semantics.

allows us to focus on the properties of the negotiation strategies rather than on explanations of our handling variables, negation, graph semantics, and determining whether a credential appears in a formula.

9 Extension to Disclosure Trees

After we introduce the concept of policy graphs, the goal of trust negotiation remains the same: finding a safe disclosure sequence which leads to the disclosure of the requested resource R . The only difference, from the negotiations point of view, is that instead of disclosing a credential's policy in a message, a party may disclose several policies for the same credential. Note that, according to TrustBuilder protocol, a party only informs the other party that a policy is for a certain resource R and does not reveal any relations between policies for the same resource. Therefore, during the negotiation, generally it is very hard for a party to collect complete information about the other party's policy graphs. (We say "generally" because when a policy graph is very simple, for example, a chain, then in the process of trust negotiation, by observing a party's messages, one may get complete structural information of a resource's policy graph.) When a party discloses enough credentials to satisfy a resource's policy of the other party, it may not expect that the resource is unguarded and can be disclosed. Instead, it only means that the current layer of that resource's policy graph is satisfied. What can be disclosed later is the next layer of the resource's policy graph.

As shown in section 6.1, disclosure trees are a straight forward way to represent the current status during a trust negotiation. Here we need to extend the definition of disclosure trees to fit it into the context of policy graphs. Suppose R is a credential or service and G_r is R 's policy graph. Then all the nodes except the sink node represent policies. We call them *policy nodes*. We assume that each policy node has a unique id (which can either be assigned by the owner $\mathcal{P}$ of the credential or by the other participants when it receives policies from $\mathcal{P}$). Therefore, given a policy node n , we can tell which resource's policy graph n belongs to.

Definition 9.1. *A graph disclosure tree (disclosure tree for short when the context is clear) for R is a finite tree satisfying the following conditions:*

1. *The root represents R .*
2. *Except for the root, each node represents either a credential or a policy. When the context is clear, we refer to a credential node (resp. policy node) by the name of the credential (resp. id of the policy) it represents. We also consider the root R as a credential node.*
3. *A credential node C is either a leaf node or it has only one child which is a policy node N_p such that N_p appears in C 's policy graph.*
4. *A policy node N_p is always an internal node whose children are all credential nodes. N_p 's children form a minimal solution set to the policy represented by N_p .*

Given a disclosure tree T , if there is a credential appearing twice in the path from a leaf node to the root, then we call T a redundant disclosure tree.

Comparing with definition 6.1, we introduce policy nodes into disclosure trees which reflect the relationship between a policy and a resource's policy graph.

Definition 9.2. Let R be a resource with policy graph G_r . Given a set S_c of credentials, if there is a path from the source to the sink node R in G_r such that each policy node in the path is satisfied by S_c , then we say S_c is a solution set to R . Further, if any proper subset of S_c is not a solution set to R , then we say S_c is a minimal solution set to R .

Definition 9.3. Given a set S_c of credentials with denial policies, we say a resource R is unsolvable regarding S_c if

1. R is with a denial policy; Or
2. Every minimal solution set to R contains at least one credential in S_c .
3. every minimal solution set to R contains at least one credential which is unsolvable regarding S_c .

As discussed in previous sections, when a resource is protected by only a single policy, a denial policy is sent only if a resource is not available (i.e., a party does not have that resource or does not want to disclose it to others under any circumstances). This requirement is valid because once a resource R 's policy is disclosed, by checking all the unavailable resources that are already known, the other party will know for sure whether R can be possibly unlocked. For example, suppose R 's policy is $R \leftarrow C_1 \wedge C_2$ and R is held by $\mathcal{P}_A$. If $\mathcal{P}_B$ does not possess credential C_1 , then after receiving R 's policy, $\mathcal{P}_B$ will know for sure that R 's policy can not be satisfied. However, in the context of policy graphs, a resource may be associated with several policies. A party does not need to disclose all its unlocked policies to the other party. Therefore, during trust negotiation, even all the disclosed unlocked policies of a resource R are eventually unable to be satisfied, the other party can not conclude that resource R can not be unlocked. So, in order to inform the other party that a resource R can not be unlocked, a party has to send a denial policy of R .

Similar to section 6.1, we are going to define some operations on extended disclosure trees.

Definition 9.4. Given a disclosure tree T and a set of S_c of credentials, the reduction of T by S_c , $\text{reduction}(T, S_c)$, is the disclosure tree T' which is obtained by removing all the subtrees rooted at a node representing resource $C \in S_c$ and all the subtrees rooted at a policy node that is satisfied by S_c . Given a set S_t of disclosure trees, $\text{reduction}(S_t, S_c) = \{\text{reduction}(T, S_c) \mid T \in S_t\}$.

Definition 9.5. Given a disclosure tree T and a policy set S_p containing no denial policies, the expansion of T by S_p , $\text{expansion}(T, S_p)$, is the set of all disclosure trees T_i such that

1. T is a subgraph of T_i , i.e., there exists a set S of credentials such that $\text{reduction}(T_i, S) = T$.
2. For each edge (P, C) in T_i such that P is a policy node and C is a credential node, if (P, C) is not an edge of T , then P is a policy in S_p .
3. For each edge (C, P) of T_i such that P is a policy node and C is a credential node, if (C, P) is not an edge of T , then P is in S_p and P is a policy node in C 's policy graph.
4. For each leaf node C of T_i , either S_p does not contain a policy which is a policy node in C 's policy graph, or T_i is redundant.

Given a set of disclosure trees S_t , $\text{expansion}(S_t, S_p) = \bigcup_{T \in S_t} \text{expansion}(T, S_p)$.

Definition 9.6. Given a set S_t of disclosure trees and a set S_{dp} of denial policies, the denial pruning of S_t by S_{dp} , denoted $\text{prune}_{\text{denial}}(S_t, S_{dp})$, is the set

$$\{T \mid T \in S_t \text{ and } T \text{ contains no resource whose policy is in } S_{dp}\}.$$

Definition 9.7. Given a set S_t of disclosure trees, the redundancy pruning of S_t , denoted $\text{prune}_{\text{redundant}}(S_t)$, is the set

$$\{T \mid T \in S_t \text{ and } T \text{ is not a redundant disclosure tree}\}.$$

Definition 9.8. Given a disclosure tree T and a set S_{dp} of denial policies, S_p of non-denial policies, and S_c of credentials, let $S = S_{dp} \cup S_p \cup S_c$. The evolution of T by S , denoted $\text{evolution}(T, S)$, is

$$\text{prune}_{\text{redundant}}(\text{prune}_{\text{denial}}(\text{reduction}(\text{expansion}(T, S_p), S_c), S_{dp})).$$

Given a set S_t of disclosure trees, $\text{evolution}(S_t, S) = \bigcup_{T \in S_t} \text{evolution}(T, S)$. As a special case, when T is the disclosure tree containing only a root node R , then we say $\text{evolution}(T, S)$ is the view of S , denoted $\text{view}(S)$.

Definition 9.9. Given a set S_t of disclosure trees and a set S_{dp} of denial policies, the denial pruning of S_t by S_{dp} , denoted $\text{prune}_{\text{denial}}(S_t, S_{dp})$, is the set

$$\{T \mid T \in S_t \text{ and } T \text{ contains no resource whose policy is in } S_{dp}\}.$$

Definition 9.10. Given a set S_t of disclosure trees, the redundancy pruning of S_t , denoted $\text{prune}_{\text{redundant}}(S_t)$, is the set

$$\{T \mid T \in S_t \text{ and } T \text{ is not a redundant disclosure tree}\}.$$

Definition 9.11. Given a disclosure tree T and a set S_{dp} of denial policies, S_p of non-denial policy nodes, and S_c of credentials, let $S = S_{dp} \cup S_p \cup S_c$. The evolution of T by S , denoted $\text{evolution}(T, S)$, is

$$\text{prune}_{\text{redundant}}(\text{prune}_{\text{denial}}(\text{reduction}(\text{expansion}(T, S_p), S_c), S_{dp})).$$

Given a set S_t of disclosure trees, $\text{evolution}(S_t, S) = \bigcup_{T \in S_t} \text{evolution}(T, S)$. As a special case, when T is the disclosure tree containing only a root node R , then we say $\text{evolution}(T, S)$ is the view of S , denoted $\text{view}(S)$.

Since a disclosure tree's leaves are always credential nodes, the definition of evolvable leaves for a negotiation party is still valid.

The intuition behind the above operations is quite similar to that of section 6.1. Therefore we are not going to give further explanation in the rest of the paper.

10 Strategy Families for Access Control Policy Graphs

In this section, we are going to present a strategy family for trust negotiation when resources are protected by policy graphs. Throughout this section, we assume that $G = (m_1, \dots, m_k)$ is a sequence of messages such that $m_i \neq \emptyset$ and $R \notin m_i$ for $1 \leq i \leq k$. We assume L_A and L_B are the local policies graphs of party $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively, and $S_d = \bigcup_{1 \leq i \leq k} m_i$. Without loss of generality, we assume $\mathcal{P}_A$ will send the next message to $\mathcal{P}_B$.

Definition 10.1. The Disclosure Tree Strategy for policy graphs (DTSG for short) is a strategy $\text{DTSG}(G, L_A, R)$ such that:

1. $DTSG(G, L_A, R) = \{\emptyset\}$ if and only if R is held by $\mathcal{P}_A$ and R is unsolvable regarding S_c where $S_c = \{C \mid \text{There is a denial policy for } C \text{ in } S_d\}$ or $\text{view}(S_d)$ has no evolvable trees for $\mathcal{P}_A$.
2. Otherwise, $DTSG(G, L_A, R)$ contains all messages m' such that one of the following conditions holds:
 - $m' = \{R\}$, if R is unlocked by credentials in S_d ;
 - m' is a non-empty set of credentials and policies such that $\text{view}(S_d \cup m')$ contains at least one evolvable tree for $\mathcal{P}_B$, and no non-empty proper subset of m' has this property.

Theorem 10.1. *The set of strategies generated by DTSG is a family.* □

Theorem 10.2. *If a strategy f and DTSG are compatible, then $f \succeq DTSG$.* □

We call the family generated by DTSG the *DTSG family*.

Corollary 10.1. *The DTSG family is closed.* □

Theorem 10.3. *Let f_1 and f_2 be strategies in the DTSG family and let f' be a hybrid of f_1 and f_2 . Then f' is also in the DTS family.* □

11 Summary and Future Work

Instead of proposing another strategy for automated trust negotiation, this paper focuses on guaranteeing interoperability between different strategies. We first propose a very simple trust negotiation protocol for the TrustBuilder trust negotiation architecture. Then we study strategies that adhere to this protocol. We introduce the concepts of strategy families and closed sets of strategies. If two strategies are in the same strategy family, then they will always correctly interoperate with each other. Closure expresses the maximality of a strategy family, i.e., if we add another strategy to a closed family, the resulting set of strategies is no longer a family. In practice, we want to identify closed families of strategies because they give negotiation participants maximum freedom in choosing the strategies appropriate for them. We introduce the concept of disclosure trees and identify the natural mapping between full disclosure trees and safe credential disclosure sequences. We then propose a strategy called the disclosure tree strategy (DTS), and prove that all the strategies that are no more cautious than DTS form a closed strategy family. We also give examples of practical strategies from the DTS family.

Further, by giving informative examples, we show that, in some situations, a policy itself may also contain sensitive information and need to be protected by unauthorized access. We introduced the concept of policy graphs to handle such situations. We extended the concept of disclosure trees such that it can reflect the relationship between resources and their policy nodes. Therefore, the DTS family is naturally extended for trust negotiation where resources are protected by policy graphs instead of a single policy.

We are currently investigating strategy families for use with non-propositional policy languages which is more expressive. We are also implementing TrustBuilder for testbed experimentation in e-commerce applications, and investigating more sophisticated definitions of “minimal” disclosure for use with practical policies.

References

- [1] K. R. Apt, D. S. Warren, and M. Truszczyński (editor). *The Logic Programming Paradigm: A 25-Year Perspective*. Springer-Verlag, 1999.
- [2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System Version 2. In *Internet Draft RFC 2704*, September 1999.
- [3] M. Blaze, J. Feigenbaum, and A. D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. In *Security Protocols Workshop*, Cambridge, UK, 1998.
- [4] P. Bonatti and P. Samarati. Regulating Service Access and Information Release on the Web. In *Conference on Computer and Communications Security*, Athens, November 2000.
- [5] T. Dierks and C. Allen. The TLS Protocol Version 1.0. In <http://www.ietf.org/rfc/rfc2246.txt>, January 1999.
- [6] S. Farrell. TLS Extension for Attribute Certificate Based Authorization. In <http://www.ietf.org/internet-drafts/draft-ietf-tls-attr-cert-01.txt>, August 1998.
- [7] A. Frier, P. Karlton, and P. Kocher. *The SSL 3.0 Protocol*. Netscape Communications Corp., November 1996.
- [8] A. Herzberg, J. Mihaeli, Y. Mass, D. Naor, and Y. Ravid. Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2000.
- [9] <http://www.ietf.org/html.charters/spki-charter.html>. *Simple Public Key Infrastructure (SPKI)*.
- [10] International Telecommunication Union. *Rec. X.509 - Information Technology - Open Systems Interconnection - The Directory: Authentication Framework*, August 1997.
- [11] N. Islam, R. Anand, T. Jaeger, and J. R. Rao. A Flexible Security System for Using Internet Content. *IEEE Software*, 14(5), September 1997.
- [12] W. Johnson, S. Mudumbai, and M. Thompson. Authorization and Attribute Certificates for Widely Distributed Access Control. In *IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 1998.
- [13] K. Seamons, M. Winslett, and T. Yu. Limiting the Disclosure of Access Control Policies during Automated Trust Negotiation. In *Network and Distributed System Security Symposium*, San Diego, CA, April 2001.
- [14] W3C, <http://www.w3.org/TR/WD-P3P/Overview.html>. *Platform for Privacy Preferences (P3P) Specification*.
- [15] W. Winsborough, K. Seamons, and V. Jones. Negotiating Disclosure of Sensitive Credentials. In *Second Conference on Security in Communication Networks*, Amalfi, Italy, September 1999.
- [16] T. Yu, X. Ma, and M. Winslett. PRUNES: An Efficient and Complete Strategy for Automated Trust Negotiation over the Internet. In *Conference on Computer and Communication Security*, Athens, Greece, November 2000.

- [17] P. Zimmerman. *PGP User's Guide*. MIT Press, 1994.

Appendix

A Proofs of Theorems

Proof of theorem 6.1: By induction on n .

When $n = 1$, resource R is unprotected. We can have a disclosure tree with only the root node, and the theorem holds.

Assume when $n \geq k \geq 1$, the theorem holds.

When $n = k + 1$, since resource R is eventually granted, we can find a minimal solution set $\{C_{j_1}, \dots, C_{j_t}\}$ for R among the credentials in G . Since G is a non-redundant safe disclosure sequence, for each C_{j_i} , where $1 \leq i \leq t$, $(C_1, \dots, C_{j_i})$ is a safe disclosure sequence with length no more than k . By the induction hypothesis, there is a non-redundant disclosure tree T_i whose root is C_{j_i} and all the nodes are credentials appearing in $(C_1, \dots, C_{j_i})$, which is a prefix of G . And for every credential pair (C'_1, C'_2) such that C'_1 is an ancestor of C'_2 in T_i , C'_2 is disclosed before C'_1 in $(C_1, \dots, C_{j_i})$. By adding R as the root and all the roots of T_i , $1 \leq i \leq t$, as children of R , we get a full non-redundant disclosure tree that satisfies conditions 1) and 2) in the theorem. Therefore the theorem holds. $\square$

Proof of theorem 6.2: Let $G = (C_1, \dots, C_n = R)$ be the post-order traversal of T . According to the definition of full disclosure trees, when a credential is disclosed in G , either it is unprotected or one of its minimal solution sets has been disclosed. Therefore its disclosure is safe, and G is a safe disclosure sequence. For every C_i in G , if there exists a credential disclosure C_j such that $j < i$ and $C_j = C_i$ in G , then we remove C_i from G . The resulting disclosure sequence is safe and non-redundant. $\square$

Proof of theorem 6.3: By induction on n , the number of nodes in T . When $n = 1$, with only a root node, T is redundant.

When $n = 2$, because T is a redundant tree, it must have only one edge (R, R) . Since T is a full tree, R is unprotected. So the tree with only the root node R is a non-redundant full disclosure tree, and T is not a legal disclosure tree.

Assume when $n = k$, where $k \geq 2$, the theorem holds.

When $n = k + 1$, let credential C' appear more than once in the path from the root to a leaf node. Suppose the path is $(R, C_1, \dots, C_i = C', \dots, C_j = C', \dots, C_k)$ such that C_i and C_j are the first and last appearance of C' in the path respectively. Since T is a full disclosure tree, the subtree rooted at C_j is also a full tree. We replace the subtree rooted at C_i by the one rooted at C_j . The resulting disclosure tree is still full but with at most k nodes. By the induction hypothesis, there is a full disclosure tree T' that is not redundant. $\square$

Proof of theorem 7.1: Suppose $\mathcal{P}_A$ and $\mathcal{P}_B$ adopt strategies f_1 and f_2 respectively, and f_1 and f_2 belong to $\text{StraSet}(DTS)$. We need to prove that if the negotiation fails, there is no safe disclosure sequence leading to the grant of access to the originally requested resource R .

When the negotiation fails, without loss of generality, we assume it is $\mathcal{P}_A$ that sends the failure message. Suppose that before $\mathcal{P}_A$ sends the failure message, $G = (m_1, \dots, m_k)$ is the sequence of exchanged messages. Therefore m_k is the last message in G that $\mathcal{P}_B$ sent to $\mathcal{P}_A$. Let L_A and L_B be the local policies of $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively, and $S_d = \bigcup_{1 \leq i \leq k} m_i$. Since $\emptyset \in f_1(G, L_A, R)$ and $f_1 \succeq DTS$, we must have $DTS(G, L_A, R) = \{\emptyset\}$. According to definition 7.1, one of the following must be true:

1. $view(S_d \cup L_A) = \emptyset$.
2. $view(S_d)$ has no evolvable tree for $\mathcal{P}_A$.

When 2) holds but 1) does not hold, we must have

$$DTS((m_1, \dots, m_{k-1}), L_B, R) = \{\emptyset\}.$$

Otherwise, since $f_2 \succeq DTS$ and $R \notin m_k$, m_k must be a superset of a minimal set m' of credentials and policies such that $view((S_d - m_k) \cup m')$ has at least one evolvable tree for $\mathcal{P}_A$. Therefore, $view(S_d)$ also has at least one evolvable tree for $\mathcal{P}_A$, which contradicts 2). Since $DTS((m_1, \dots, m_{k-1}), L_B, R) = \{\emptyset\}$, that means that before $\mathcal{P}_B$ sent m_k , one of 1) or 2) was true. Since at the beginning of the negotiation, 2) is not satisfied, we know that in some previous stage of the negotiation, 1) must have been true.

When 1) holds, that means no tree will evolve to be a full disclosure tree. So there is no safe disclosure sequence leading to the grant of access to R . $\square$

Proof of theorem 7.2: By contradiction.

Suppose that $\mathcal{P}_A$ is the local party. Assuming $DTS \not\preceq f$, then there exists a choice of G , L_A and R such that

$$\exists m' \in f(G, L_A, R) \text{ such that } \forall m \in DTS(G, L_A, R), m \not\subseteq m'$$

Then the following must be true:

- $view(S_d \cup L_A) \neq \emptyset$.
- $view(S_d)$ has at least one evolvable tree for $\mathcal{P}_A$.

Otherwise, according to definition 7.1, $DTS(G, L_A, R) = \{\emptyset\}$ and $\emptyset \subseteq m'$. Also, we have $R \notin m'$. Otherwise, R is unlocked by S_d and $\{R\} \in DTS(G, L_A, R)$. $\{R\} \subseteq m'$.

$DTS(G, L_A, R)$ contains all the minimal sets m of local policies and credentials such that $view(S_d \cup m)$ has at least one evolvable tree for $\mathcal{P}_B$. Since $DTS \not\preceq f$, $view(S_d \cup m')$ has no evolvable tree for $\mathcal{P}_B$. Thus, after sending m' , DTS at $\mathcal{P}_B$ will send a failure message and end the negotiation. However, since $view(S_d \cup L_A) \neq \emptyset$, there is a tree in $view(S_d \cup L_A)$ whose leaves $\{C_1, \dots, C_t\}$ are all evolvable for $\mathcal{P}_B$ and none of those credentials' policies is in S_d . So if all those credentials are unprotected, then there exists a full disclosure tree, which means that f and DTS are not compatible, leading to a contradiction. $\square$

Proof of theorem 7.3: $\forall G, L, R$, let $f'(G, L, R) = \{m_1, \dots, m_k\}$. Since f' is a hybrid of f_1 and f_2 , $m_i \in f_1(G, L, R)$ or $m_i \in f_2(G, L, R)$, for all $1 \leq i \leq k$. Because both f_1 and f_2 are in the DTS family, there exists $m' \in DTS(G, L, R)$ such that $m' \subseteq m_i$. Thus, $f' \succeq DTS$. $\square$

Proof of theorem 7.4: Because TrustBuilder-Simple $\succeq$ TrustBuilder-Relevant, it suffices to show that $DTS \preceq$ TrustBuilder-Relevant.

Suppose $\mathcal{P}_A$ is the local party and $TrustBuilder-Relevant(G, L_A, R) = \{m\}$. If $m = \emptyset$, that means $\mathcal{P}_A$ has no relevant policies and unlocked credentials other than those in S_d . Then $view(S_d)$ must have no evolvable trees for $\mathcal{P}_A$. Otherwise, suppose $T \in view(S_d)$ has an evolvable leaf C for $\mathcal{P}_A$. Then, neither C nor C 's policy is in S_d and C is syntactically relevant to R . If C is unlocked by credentials in S_d , then $C \in m$. Otherwise, C 's policy is in m . Both cases contradict the assumption that $m_r = \emptyset$.

When $m \neq \emptyset$, if $DTS(G, L_A, R) = \{\emptyset\}$, then $\emptyset \subseteq m$. Otherwise, according to definition 7.1, we must have $view(S_d \cup L_A) \neq \emptyset$ and $view(S_d)$ contains at least one evolvable tree for $\mathcal{P}_A$. Suppose $m = \{d_1, \dots, d_t\}, 1 \leq t$. If $view(S_d)$ already has an evolvable tree for $\mathcal{P}_B$, then $\{d_1\}$ is a non-empty minimal set such that $view(S_d \cup \{d_1\})$ has an evolvable tree for $\mathcal{P}_B$. So $\{d_1\} \in DTS(G, L_A, R)$ and $\{d_1\} \subseteq m$. On the other hand, if $view(S_d)$ has no evolvable tree for $\mathcal{P}_B$, consider any $m' \in DTS(G, L_A, R)$. Let d be any disclosure in m' . If d is the disclosure of a credential C , then C must appear in a disclosure tree in $view(S_d \cup (m' - \{d\}))$. Otherwise, $view(S_d \cup (m' - \{d\})) = view(S_d \cup m')$ which contradicts the fact that m' is a minimal set. By similar arguments, when d is a disclosure of credential C 's policy, we also have C appearing in a disclosure tree in $view(S_d \cup (m' - \{d\}))$. By proposition 7.1, C is syntactically relevant to R . Therefore, $d \in m$, which means $m' \subseteq m$. Thus, $DTS \preceq \text{TrustBuilder-Relevant}$. $\square$

Proof of theorem 7.5: For TrustBuilder-Simple, every time there are new credential disclosures in an incoming message, TrustBuilder-Simple needs to scan each resource's policy and check whether it is unlocked. Since there are at most n credential disclosures, such cost is bounded by $O(nm)$. To determine when to disclose a denial policy, TrustBuilder-Simple only needs to scan each incoming policy disclosure once, at a cost of no more than $O(m)$. Therefore, the total computation cost of TrustBuilder-Simple during trust negotiation is bounded by $O(nm)$.

Compared to TrustBuilder-Simple, besides the cost for checking whether resources are unlocked and when to disclose a denial policy, the only extra cost of TrustBuilder-Relevant is to determine whether a credential is relevant to R . At the beginning of the negotiation, R is the only known resource that is relevant to R . During the negotiation, every time there is a policy disclosure of a resource relevant to R , TrustBuilder-Relevant can scan the policy and mark all the credentials appearing in it as relevant. Similarly, every time TrustBuilder discloses a policy of a relevant resource, it can also mark those credentials appearing in the policy as relevant. By this way, in the whole negotiation process, all the policies are scanned only once to determine relevant resources. Therefore the cost is $O(m)$. Thus, the total computation cost of TrustBuilder-Relevant is also bounded by $O(nm)$. $\square$

Proof of theorem 10.1: Suppose $\mathcal{P}_A$ and $\mathcal{P}_B$ adopt strategies f_1 and f_2 respectively, and f_1 and f_2 belong to $\text{StraSet}(DTS)$. We need to prove that if the negotiation fails, there is no safe disclosure sequence leading to the grant of access to the originally requested resource R .

When the negotiation fails, without loss of generality, we assume it is $\mathcal{P}_A$ that sends the failure message. Suppose that before $\mathcal{P}_A$ sends the failure message, $G = (m_1, \dots, m_k)$ is the sequence of exchanged messages. Therefore m_k is the last message in G that $\mathcal{P}_B$ sent to $\mathcal{P}_A$. Let L_A and L_B be the local policy graphs of $\mathcal{P}_A$ and $\mathcal{P}_B$ respectively, and $S_d = \bigcup_{1 \leq i \leq k} m_i$. Since $\emptyset \in f_1(G, L_A, R)$ and $f_1 \succeq DTSG$, we must have $DTSG(G, L_A, R) = \{\emptyset\}$. According to definition 10.1, one of the following must be true:

1. R is unsolvable regarding S_c where $S_c = \{C | \text{There is a denial policy for } C \text{ in } S_d\}$.
2. $view(S_d)$ has no evolvable tree for $\mathcal{P}_A$.

When 2) holds but 1) does not hold, we must have $DTSG((m_1, \dots, m_{k-1}), L_B, R) = \{\emptyset\}$. Otherwise, since $f_2 \succeq DTSG$ and $R \notin m_k$, m_k must be a superset of a minimal set m' of credentials and policies such that $view((S_d - m_k) \cup m')$ has at least one evolvable tree for $\mathcal{P}_A$. Therefore, $view(S_d)$ also has at least one evolvable tree for $\mathcal{P}_A$, which contradicts 2). Since $DTSG((m_1, \dots, m_{k-1}), L_B, R) = \{\emptyset\}$, that means that before $\mathcal{P}_B$ sent m_k , one of 1) or 2) was true. Since at the beginning of the

negotiation, 2) is not satisfied, we know that in some previous stage of the negotiation, 1) must have been true.

When 1) holds, that means R is unsolvable. So there is no safe disclosure sequence leading to the grant of access to R . $\square$

Proof of theorem 10.2: By Contradiction.

Suppose that $\mathcal{P}_A$ is the local party. Assume $DTSG \not\preceq f$, then there exists a choice of G , L_A and R such that

$$\exists m' \in f(G, L_A, R) \text{ such that } \text{forall } m \in DTSG(G, L_A, R), m \not\subseteq m'$$

Then the following must be true:

- R is not unsolvable regarding $S_c = \{C | \text{There is a denial policy of } C \text{ in } S_d\}$.
- $\text{view}(S_d)$ has at least one evolvable tree for $\mathcal{P}_A$.

Otherwise, according to definition 10.1, $DTSG(G, L_A, R) = \{\emptyset\}$ and $\emptyset \subset m'$. Also, we have $R \not\subseteq m'$. Otherwise, R is unlocked by S_d and $\{R\} \in DTSG(G, L_A, R)$. $\{R\} \subset m'$.

$DTSG(G, L_A, R)$ contains all the minimal sets m of local policies and unlocked credentials such that $\text{view}(S_d \cup m)$ has at least one evolvable tree for $\mathcal{P}_B$. Since $DTSG \not\preceq f$, $\text{view}(S_d \cup m')$ has no evolvable tree for $\mathcal{P}_B$. Thus, after sending m' , $DTSG$ at $\mathcal{P}_B$ will send a failure message and end the negotiation. However, since R is not unsolvable regarding S_c , there is a minimal solution set to R which contains no unsolvable credentials regarding S_c . Therefore, if all the credentials are actually freely available, then there exists a sequence of credential disclosures leading to the disclosure of R , which means f and $DTSG$ are not compatible, leading to a contradiction. $\square$

B An Example of Trust Negotiation

Consider an online medical records service that allows patients to access their electronic medical records from Busey Hospital over the web. In our example, a parent wants to access her child's medical records at the online service. The policies of the parent and the online service are shown in figure 9. Table 1 gives the interpretations of the credentials appearing in figure 9.

This example fits most naturally into a simple first-order policy language, such as a datalog program. To be in keeping with the rest of this paper, we have fitted the example into a propositional format. This hides much of the example's natural complexity behind the translation of credentials to propositional symbols. In particular, several of the individual propositional symbols (S_2, S_3, C_2, C_3, C_5 and C_6) in table 1 are best represented by credential chains, where the issuer of one credential is the owner of the next credential in the chain. For example, a birth certificate is issued by a county clerk, who is certified by her county to act as a county clerk. In turn, each county is certified by the state, possession, or territory in which it is located. In turn, the states/possessions/territories are certified by the Federal government, which is at the top of this particular set of credential chains. Similarly, the distinction between an adult's and a child's birth certificates is most easily expressed in a first-order policy language.

Further, authentication of the requester to one of the principals mentioned in the credentials is a key aspect of many of the policies in figure 9, but is not included in this propositional format. For example, in the policy $R \leftarrow (C_1 \wedge C_2) \vee (C_7 \wedge (C_3 \vee C_5 \vee C_6)) \vee C_4$, to satisfy the first clause of the policy, the requester must authenticate to the owner of C_1 and C_2 , and the owner of C_1 must be the patient whose records are being requested. To satisfy the second clause, the requester

must authenticate to one of the parents in the child's birth certificate, or the guardian in the guardian credential, or one of the parents in the adoption credential. Further, the child named in the adoption/guardianship/birth certificate must be the owner of the child patient ID, and must be the patient whose records have been requested. To satisfy the third clause, the requester must authenticate to the owner of the employee ID.

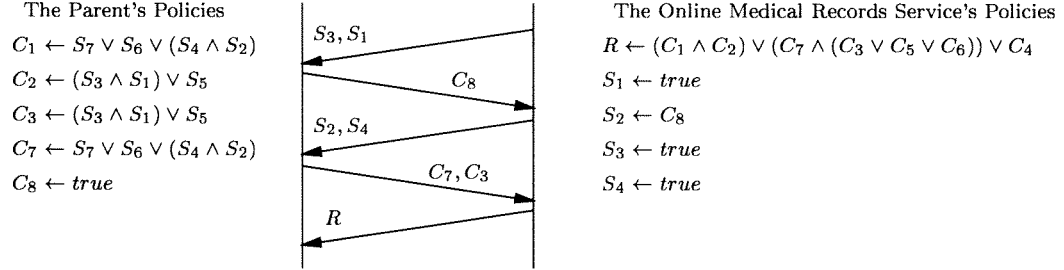


Figure 9: An example of access control policies and a safe disclosure sequence.

<i>Credential</i>	<i>Interpretation</i>
R	patient's medical records from Busey Hospital, stored at the Online Medical Records Service
S_1	membership credential issued by TrustE
S_2	mailing address certificate issued by the US Post Office, showing an address in Illinois
S_3	accreditation credential issued by the US Government to medical facilities
S_4	affiliate organization credential issued by Busey Hospital
S_5	US Government agency credential issued by the US Government
S_6	employee ID issued by Blue Cross Blue Shield of Illinois
S_7	employee ID issued by Busey Hospital
C_1	adult patient ID credential issued by Busey Hospital
C_2	adult's birth certificate (age over 21)
C_3	child's birth certificate (age under 21)
C_4	employee ID issued by Busey Hospital to a medical practitioner
C_5	legal guardianship credential issued by a state court
C_6	adoption credential issued by a state court
C_7	child patient ID credential issued by Busey Hospital
C_8	membership credential issued by the Reduce Junk Mail Alliance

Table 1: Interpretations of credentials appearing in figure 9.