PRUNES: An Efficient and Complete Strategy for Automated Trust Negotiation over the Internet

Ting Yu Xiaosong Ma Marianne Winslett
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W.Springfield Ave.
Urbana, IL 61801

{tingyu, xma1, winslett}@cs.uiuc.edu

ABSTRACT

The Internet provides an environment where two parties, who are virtually strangers to each other, can make connections and do business together. Before any actual business starts, a certain level of trust should be established. Each party should make sure that the other one is qualified and can be trusted for the ongoing business. Property-based digital credentials [1] make it possible to prove that a party satisfies certain requirements imposed by the ongoing business. The problem is that digital credentials themselves also contain valuable information which a party does not want to show to just any strangers. Therefore, for each credential there is usually a disclosure policy associated with it, indicating under what circumstances this credential can be disclosed. An automated trust negotiation strategy needs to be adopted to establish trust between two parties based on their disclosure policies. Previously proposed negotiation strategies may either fail when in fact success is possible, disclose irrelevant credentials, or have a high communication complexity. In this paper, we present a trust negotiation strategy, Prudent Negotiation Strategy (PRUNES), that guarantees that trust is established, if allowed by the credential disclosure policies. Meanwhile PRUNES makes sure that no irrelevant credentials are disclosed during trust negotiations. We also prove that PRUNES is efficient: in the worst case, the communication complexity is $O(n^2)$ and the computational complexity is $O(nm)$, where n is the number of credentials and m is the size of the credential disclosure policies in disjunctive normal form.

1. INTRODUCTION

In a traditional distributed environment, service providers and requesters are usually known to each other. Often shared information in the environment tells which parties can provide what kind of service and which parties can get those services. Thus, trust between parties is not a problem. Even if in some occasions there is a trust issue, like in traditional client-server systems, the trust establishment is usually one-directional in the sense that the server is a well-known service provider. Before requesting any service, the client must prove it is qualified for that service. In this case, trust establishment is often handled by uni-directional access control methods. For example, the client must log on as a pre-registered user.

One of the Internet's significant differences from traditional distributed systems is that the Internet provides an environment where different parties may make connections and do business without being previously known to each other. In many cases, before any actual business starts, a certain level of trust should be established from scratch. Generally, trust establishment is done through exchange of information between the two sides. Since both parties are not known to each other, this trust establishment process should be bi-directional: both sides may have sensitive information which they are reluctant to disclose, until the other side has proved to be trustworthy at a certain level. As there are more service providers emerging on the web every day, and people are performing more sensitive transactions (for example, financial and health services) via the Internet, this need for building mutual trust will become more and more common.

However, currently on the Internet, the prevailing approach is still the traditional access control method, where customers are required to create their own accounts and log on before getting any service. Customers often have to, *unconditionally*, provide sensitive information in creating their accounts, while there is no easy way for them to check the qualifications or reliability of the service provider. Further, this method creates a large management overhead for both sides. The service provider must maintain a large database of customer profiles. The customer must remember the user name and password for accounts from many service providers. Besides the management headache, the tedious process of user registration often scares away customers who just want to take one minute to do things like checking an air ticket price to an unusual destination, or posting a message on a web board they do not visit frequently, thereby reducing potential business for the service provider.

What's more, the traditional access control method usually requires the customer to reveal her exact identity, which, in many cases, is irrelevant to the service the customer requests. For example, suppose an on-line digital library con-

CCS '00, Athens, Greece.

tains books, journals and conference proceedings that can be divided into three categories: generally accessible, accessible to ACM members only and accessible to IEEE members only. When a user wants to read a paper in ACM Transactions on Information and System Security, it suffices to prove that she is actually a member of ACM. Other personal information like phone numbers and home address does not need to be revealed. In some situations, revealing personal identification is highly undesirable. For example, a user may query an on-line medical database about a certain disease. If the identification of the user is revealed, it's very reasonable to assume that the user herself or some relative of the user may have that disease, which may be highly sensitive information.

Property-based *digital credentials* [1] (or simply *credentials*) make it feasible to manage trust establishment efficiently and bi-directionally on the Internet. Digital credentials are the on-line counterparts of paper credentials that people use in their daily life, such as a driver's license. They are signed by authorized parties and can be made verifiable and unforgeable. By showing necessary credentials to each other, the service requester and provider can both prove their qualifications. When credentials do not bear sensitive information, the trust establishment procedure is very simple. For example, in the on-line library mentioned above, the user (or a software agent acting on behalf of the user, such as a browser) first checks the library's qualification credentials, which are always available. Then she presents her ACM membership credential along with the retrieval request. However, in many situations, especially in the context of e-business, credentials themselves carry some sensitive information. 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, a more complex procedure needs to be adopted to establish trust through negotiation. Each Internet entity can have its own *security agent* (SA) to be in charge of negotiations and revealing credentials. It is highly desirable that the negotiations be automatic, that is, given user credentials and policies guiding the disclosure of these credentials, negotiations with any other Internet entity can be carried out by the user's SA automatically, without the user's intervention. The security agent can adopt different *negotiation strategies* in handling negotiations. For example, it can be *parsimonious*, always trying to avoid disclosing local credentials as much as possible, or *eager*, always trying to get or grant the service as soon as possible without worrying much about credential disclosure [10].

In this paper, we present PRUNES, a trust negotiation strategy which guarantees to succeed whenever trust establishment is possible between two parties. Meanwhile PRUNES makes sure that no credential will be disclosed if the negotiation fails, and no irrelevant credentials will be disclosed if the negotiation succeeds. We also prove that PRUNES is efficient: in the worst case, the communication complexity is $O(n^2)$ and the computational complexity is $O(nm)$, where n is the number of credentials involved in the trust establishment and m is the total size of the credential disclosure policies for these credentials, expressed in disjunctive normal form.

The paper is organized as follows. Section 2 defines credential disclosure policies and discusses properties of trust negotiation strategies. In section 3 we give the main result of the paper, an efficient complete strategy and the proof of its communication and computational complexity. Section 4 describes works related to automated trust negotiation. Section 5 draws conclusions and describes directions of possible future work.

2. CREDENTIAL DISCLOSURE POLICIES AND TRUST NEGOTIATION STRATEGIES

A *credential* is a digitally signed assertion by the *credential issuer* about the *credential owner* [1, 12]. Using modern encryption technology, the issuer signs a credential, which describes one or more attributes of the owner, by its own private key. The public key of the issuer can be used to verify that the credential is actually issued by the issuer. The signed credential also includes the public key of the owner. The owner can use her private key to authenticate herself as the owner of the credential [7]. So digitally signed credentials can be verifiable and unforgeable.

When credentials do not contain sensitive information, i.e, they can be shown to anybody whenever requested, only the service itself needs to be protected. The service requester will be willing to provide any credentials requested by the provider in order to get the service. In this case, the trust negotiation can be finished in a single round.

When credentials contain sensitive information and need to be protected, a single-round trust negotiation is no longer sufficient. Before a party is willing to disclose a credential, a certain level of trust should already be established. For example, a user filing an on-line application for an apartment may be asked for her digital social security card for trust establishment and a future credit check. She may reply that her digital social security card can be shown only if the other party has credentials to prove that it is authorized to accept such a card and that it adheres to guidelines for securing social security numbers. Therefore, the trust negotiation will require multiple rounds. More generally, the trust between two parties often will need to be built step by step by exchanging credentials before the requester can get the service. We call the constraints on the disclosure of a sensitive credential the *disclosure policy* (or simply *policy*) for the credential.

As discussed below, we will treat credentials as propositional symbols. We also assume that the credential sets of the two negotiation parties are disjoint. Formally, we define a credential disclosure policy for credential C to be in the form

$$C \leftarrow F_C(S_1, ..., S_k)$$

where $F_C(S_1, ..., S_k)$ is an expression involving only credentials from other parties, the Boolean operators $\vee$ and $\wedge$, and parentheses as needed. S_i is satisfied if and only if the other party has shown credential S_i. Credential C can be shown to the other party only if $F_C(S_1, ..., S_k)$ is satisfied. A

successful negotiation finds a credential exchange sequence $G = L_1, ..., L_j, S$, where each L_i is a credential, for $1 \leq i \leq j$, such that when two parties exchange credentials in the order defined by G, F_{L_i} evaluates to *true* when it is time for L_i to be shown to the other party. Note that the requester can get the service S only if the requester proves its qualifications by showing a combination of credentials that satisfies S's access policy, which can be expressed in the same manner as a credential disclosure policy. So we treat S as a credential too throughout this paper.

Once a credential C's disclosure policy F_C is satisfied, we say credential C is *solved*. Similarly, given a logical expression E of credentials, we say E is *solved* when E evaluates to *true*. If the disclosure of $L_1, \ldots, L_k$ will solve E, we call $L_1 \wedge \cdots \wedge L_k$ a *solution* to E. A credential C is called *unprotected* if it can be disclosed freely whenever requested. We use $C \leftarrow \epsilon$ to denote an unprotected credential's policy. An intuitive observation is that two parties cannot establish trust unless there is at least one unprotected credential on either side.

Given a credential S, we say a credential C is *syntactically relevant* to S if and only if

- C appears in S's policy, or
- C appears in the policy of a credential C' which is relevant to S.

Given a credential set P, if the disclosure of credentials in P satisfies credential S's policy, we say P is a *satisfaction set* of C. If none of P's proper subsets is a satisfaction set S, we say P is a minimal satisfaction set. If a credential C appears in one of S's minimal satisfaction sets, we say C is *semantically relevant* to S. In this paper, we assume that policies are written (or rewritten upon receipt) so that syntactic relevance always implies semantic relevance. In disjunctive normal form, this is done by removing each disjunct that is subsumed by another disjunct. For example, $C \leftarrow A \vee (A \wedge B)$ should be rewritten as $C \leftarrow A$.

The left side of Figure 1 gives an abstract example of credential policies and a credential exchange sequence which establishes trust for the client to get service S according to the client and the server's disclosure policies. Note that we assume there is no third entity which knows both sides' credential policies. A negotiation strategy is run on each side's SA, which is only aware of local disclosure policies. The right side of Figure 1 gives a realistic example, the landscape designer's purchase from CPN discussed earlier.

[10] proposed some desirable properties of negotiation strategies. A negotiation strategy should be *complete*. By complete we mean that the SAs of the two sides can always find a successful negotiation whenever the credential policies of both sides allow one. It should terminate with failure when a successful negotiation does not exist. It should not disclose any credentials that are not essential to a negotiation. Finally, a strategy should be reasonably efficient. The efficiency of a strategy includes two aspects: the communication cost and the computational cost. The communication cost includes both the total size of the messages and the total number of messages sent between the two SAs.

[10] proposed two different categories of negotiation strategies: eager and parsimonious. A eager negotiation strategy allows flooding-style negotiation, as both sides disclose a credential to the other side as soon as the policy of that credential is satisfied, therefore ensuring that a successful negotiation can be found in the minimum possible number of rounds. The disadvantage of such a strategy is that it usually results in disclosure of irrelevant credentials. For instance, a user's trust negotiation for a certain service with a digital library may end up disclosing credentials not requested by the service provider, such as her driver's license and health insurance card. At the other extreme, a parsimonious strategy will not allow credential exchanges until both parties know there exists a successful negotiation. When an incoming request for a credential is received by an SA and this credential has not been solved, an outgoing counter request is prepared according to its disclosure policy and sent out in response. Based on this principle, two parsimonious negotiation strategies are proposed in [10, 11].

The strategy in [10] is not complete and has difficulties in deciding when the negotiation should fail and stop. The strategy in [11] is complete. In this strategy, when a request E comes, the SA combines all the possible solutions into a counter request E' that will be satisfied iff E is satisfied, and sends E' back. It is also easy to decide when the strategy should halt. However, when two parties have many credentials, as well as complex disclosure policies, after several rounds of counter request exchanges, the size of the request may become huge. In the worst case, the size of a request can be exponential in the number of credentials in a negotiation, which brings exponential communication cost.

We close this section with a discussion of the utility of propositional policies. The obvious translation of most credentials into logic requires first-order constructs. For example, a credit card credential C might translate into the expression $\text{CreditCard}(C) \wedge \text{CreditCardType}(C) = \text{VISA} \wedge \text{Owner}(C) = \text{Jane\_Doe} \wedge \text{AccountNumber}(C) = 1234567890 \wedge \text{ExpirationMonth}(C) = 1 \wedge \text{ExpirationYear}(C) = 2000$. Further, the appropriate translation to be used for each credential must be well-known, as otherwise two negotiating parties will never reach agreement on whether particular credentials do satisfy a given policy. On the other hand, the details present in a first-order version of the CPN example would obscure rather than clarify those two policies, and the modeling power of first-order logic is not needed to capture the intent of the policies (no variables or constants need to be shared between the constraints on reseller's licenses, BBB membership, credit cards, and account numbers). Further, PRUNES is a negotiation strategy where parties may declare that they have a certain credential and are willing to present it under certain conditions, before those conditions have been reached. This is fine if, for example, the landscape designer merely declares that she has a valid credit card, without giving any further details; but it is unacceptable if she has to give out the first-order details of her credit card before the necessary trust has been established.

Fortunately, there are a number of ways to combine the advantages of propositional and first-order representations of policies and credentials, while avoiding the drawbacks of each. We will describe one such possibility here. The parties can use a canonical translation of the fields of a credential into first-order logic. When the modeling power of first-order logic is not needed to capture the high-level aspects of a policy, those aspects can be written using propositions (equivalent to 0-ary first-order predicates), as we did for the CPN example. The first time that one party re-

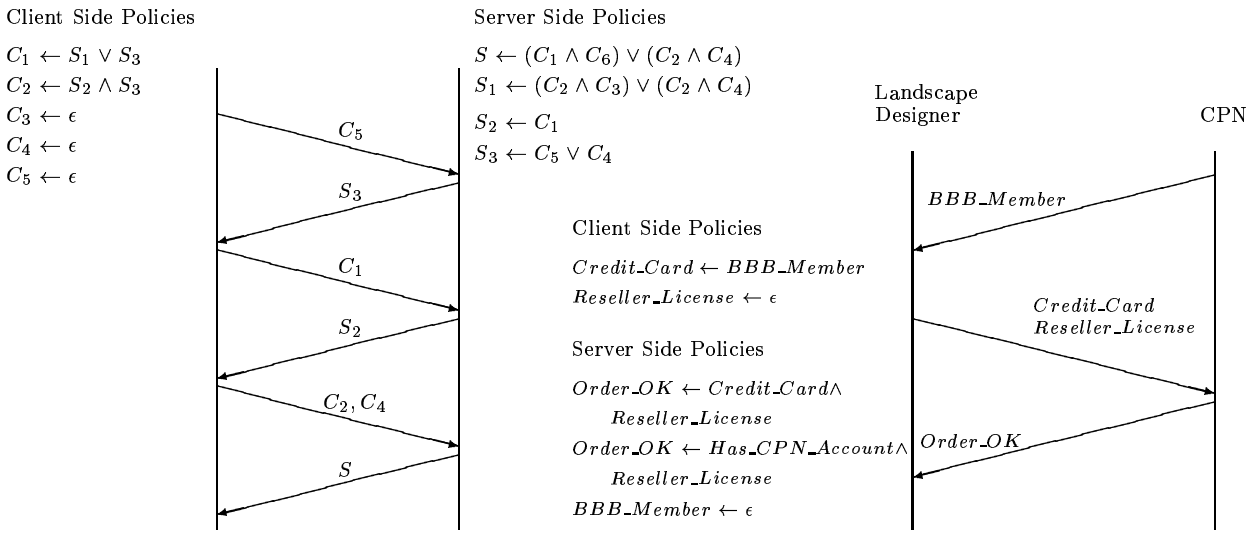

Figure 1: Two examples of disclosure policies and a credential exchange sequence which establishes trust between the server and the client.

quests a propositional "credential" C from the other party, the request can include a definition of C couched in the first-order terms used in the canonical translation, and using the same syntax as for policies. For example, when CPN asks the designer for a valid credit card (propositional symbol *Credit_Card* in Figure 1), CPN can send along a logic-programming-style definition of *Credit_Card* that abstracts away the irrelevant and/or private details of the credit card:

$$
\begin{aligned}
Credit\_Card &\leftarrow IssuerOK(x) \wedge NotExpired(x) \\
IssuerOK(x) &\leftarrow CreditCardType(C) = VISA \\
IssuerOK(x) &\leftarrow CreditCardType(C) = MasterCard \\
NotExpired(x) &\leftarrow CreditCard(x) \wedge ExpirationYear(x) > 2000 \\
NotExpired(x) &\leftarrow CreditCard(x) \wedge ExpirationYear(x) = 2000 \\
&\quad \wedge ExpirationMonth(x) > 11
\end{aligned}
$$

This definition can be used by the recipient to determine if it possesses the propositional "credential" *Credit_Card*, and to choose the correct policy to govern disclosure of *Credit_Card*, i.e., the policy used to govern disclosure of credit cards. PRUNES does not need to understand the definition of *Credit_Card* or take account of it during negotiation, so the remainder of the paper considers only purely propositional policies and propositional translations of credentials.

3. AN EFFICIENT COMPLETE STRATEGY

In this section, we first introduce a complete brute-force backtracking negotiation strategy. Then we introduce the concept of a negotiation search tree and use it to prove some properties of the backtracking search. Based on these properties, we propose PRUNES, a refined backtracking strategy that prunes certain areas of the search tree, and prove it is equivalent to the brute-force strategy in finding credential exchange sequences, therefore complete too. Finally we will show the complexity of PRUNES.

3.1 A Brute-Force Backtracking Strategy

For the rest of the paper, we assume every credential C's policy is represented in the disjunctive normal form $C \leftarrow D_1 \vee ... \vee D_l$, where $D_i = S_{i1} \wedge ... \wedge S_{ik_i}$, and each S_{ij}, for $1 \leq i \leq l$ and $1 \leq j \leq k_i$, is a credential to be shown by the opposite side. We call D_i a *clause* of C.

The strategy starts when the client's SA sends a request for service S to the server. When an SA receives a request for C, it checks C's disclosure policy. If C is an unprotected credential (or service), it is *granted* immediately, which means the SA informs the opposite side that credential C now can be solved. No actual credential exchange happens at this stage. If C does not exist, the request is denied. Otherwise, the SA tries to solve the first clause D_1 of C. Suppose $D_1 = S_1 \wedge ... \wedge S_k$. The SA will then examine each $S_i, 1 \leq i \leq k$, starting with S_1. If S_i has already been granted, the SA moves on to S_{i+1}. If the SA has already issued a request for S_i that has not been granted or denied (a *pending request*), the SA does not issue another request for S_i and S_i fails to be granted. Otherwise, the SA requests S_i from the other party. If all the credentials in D_1 are granted, then the SA knows credential C can be solved and sends a grant message for C to the opposite side. If one of the credentials in D_i cannot be granted, the SA will backtrack immediately and try D_2. If none of the clauses can be solved, a deny message for C will be sent to the opposite side.

Finally, if the request for service S is denied, then the algorithm halts and claims that successful negotiations are impossible between the two parties. Otherwise, there exists a successful negotiation. Both parties in the negotiation can now start actual credential exchange using grant and deny information accumulated during previous negotiation rounds. More details on the credential exchange procedure will be given in Section 3.2.

This brute-force backtracking strategy backtracks whenever a circular dependency is detected. Given that the sets of credentials and policies are finite at both sides, the search it conducts always terminates. Also, since it searches all the possible negotiation sequences in a depth first way, we can easily prove that it is complete.

Before we start discussing any strategy's computation and communication complexity, we introduce the concept of a tree that is traversed by trust negotiation algorithms, an AND/OR tree [9] called the *negotiation search tree*. In a negotiation search tree, there are two kinds of nodes: cre-

dential nodes and disjunctive nodes. Credential nodes correspond to credentials of both sides, while disjunctive nodes correspond to clauses of credential policies. The root of a negotiation search tree is a credential node for the service S requested by the client. If $F_C = D_1 \vee ... \vee D_n$, then every appearance of a credential node for C in the search tree will have n disjunctive nodes as its children (each of which corresponds to a clause of C's policy), except when the node for C itself is a circular dependency node, which we will define soon. Similarly, for a clause $D = C_1 \wedge ... \wedge C_k$, a disjunctive node for D will have k credential nodes as its children, each of which corresponds to a credential in D. For an unprotected credential, its credential node will have only one child, which is a special disjunctive node called a *TRUE node*. For a nonexistent credential, its credential node's only child is a special disjunctive node called a *FALSE node*. Both TRUE and FALSE nodes are leaf nodes. A credential node will be a leaf node if it is the second time it appears in the path from S to the credential node (this means a circular dependency exists and the negotiation along this path stops here). We call such a leaf node a *circular dependency node*. An SA will backtrack at a circular dependency node, without sending a request message. Also, by forcing circular dependency nodes to be leaf nodes, a negotiation search tree is always finite. Figure 2 gives an example of a negotiation tree.

It's easy to see now that the brute-force backtracking strategy is just a straight-forward depth first search of the AND/OR tree, only the search is conducted in a distributed manner by the SAs at two sites and each SA only has partial information about the tree. The two SAs move alternately in the tree, with each move triggered by an incoming message. There can be many nodes in the negotiation search tree for the same credential, each corresponding to a request for that credential at a different time during the negotiation. The size of the negotiation search tree can be exponential in n, the number of credentials at both sides, which makes the brute-force backtracking strategy's communication and computational complexity also exponential in n in the worst case.

3.2 An Efficient Complete Strategy

A key observation for the brute-force backtracking strategy is that it is not always necessary to explore a credential C's policy whenever a request for C comes. If C has just been requested a few rounds before, and the internal state of the SA has not changed much, it is very likely that the new request for C will be denied again. Now we will derive a sufficient condition for when a request for credential C will be denied, which we can use to get an efficient backtracking strategy.

In the brute-force backtracking strategy, suppose that a request R for C is denied. We will show that with the brute-force strategy, there is no way to solve C without solving one or more of the ancestors of the node for C where the denial just took place. We will further show that if, when the next request for C is received, no credential has been granted since R was denied, then not only will this new request for C be denied too, but no other credential will be granted between the time the new request is received and denied.

THEOREM 1. *In the brute-force backtracking strategy, when a request for C is denied, suppose that the path in the negotiation search tree from the root to the credential node for C corresponding to that request is $L_0, ..., L_k, C$, where L_i is a credential requested by either party, for $0 \leq i \leq k$. Then C will not be granted before at least one L_i, $0 \leq i \leq k$, is granted.*

PROOF. Proof by induction on d, the maximum distance from the node for credential C in the negotiation search tree corresponding to the denied request (for simplicity, we also call this node C for the rest of the proof), to a node N that is a proper descendent of C and is either a circular dependency node or a FALSE node. According to the conditions defined in Section 3.1 under which a request should be denied, at least one such proper descendent must exist. So $d \geq 1$.

When $d = 1$, this proper descendent must be a FALSE node (otherwise, C will not be denied immediately and d is at least 2), which means that the SA does not possess C and the theorem holds.

When $d = 2$, since the strategy must have explored all C's clauses before denying the request for C, each clause D of C's policy contains a credential among $L_0, ..., L_k$ to constitute circular dependency. In order to solve C, we must satisfy one of its clauses D. But D can only be satisfied if all of its conjuncts are solved, which means that one of $L_0, ..., L_k$ must be solved. We conclude that when $d = 2$, the theorem holds.

Suppose when $d = j$, the theorem holds.

When $d = j+1$, since C is being denied, for each clause D of C's policy, there exists a credential S' in D, which either has been denied or is a circular dependency node. If S' is a circular dependency node, then S' is identical to one of $L_0, ..., L_k, C$. Thus S' cannot be solved without solving one of $L_0, ..., L_k, C$. Otherwise, define d' for S' as we define d for C. The node for S' is two levels below the node for C in the search tree, so we have $d' \leq j - 1$. Since S' has been denied, the induction hypothesis applies to the node for S', according to which S' again cannot be be solved without solving one of $L_0, ..., L_k, C$. Because there is such an S' in every clause of C's policy, C cannot be solved without first solving one of $L_0, ..., L_k, C$, and the theorem holds. □

THEOREM 2. *In the brute-force backtracking strategy, suppose there are two requests R_1 and R_2 for C, with R_1 arriving before R_2. Let $\mathcal{G}$ be the set of granted credentials from both sides and suppose that when R_1 is denied, $|\mathcal{G}| = k$. If when R_2 arrives, we still have $|\mathcal{G}| = k$, then (1) R_2 will also be denied, and (2) when R_2 is denied, $|\mathcal{G}| = k$.*

PROOF. Suppose the paths from the root of the negotiation search tree to nodes for credential C when R_1 and R_2 are issued are $p_1 = L_{10}, ..., L_{1k}, C$ and $p_2 = L_{20}, ..., L_{2l}, C$ respectively. Consider the last common node shared by p_1 and p_2, tracking down from the root S. Such a node exists because at least $L_{10} = L_{20} = S$. More precisely, let $t = max\{j | for\ all\ i\ such\ that\ 0 \leq i \leq j,\ L_{1i} = L_{2i}\}$.

Suppose for request R_2, the search algorithm finds a credential exchange sequence E to solve C. Since $L_{10}, ..., L_{1t}$ are ancestors of C in p_2, E does not contain any of $L_{10}, ..., L_{1t}$. If E does not contain any of $L_{1t+1}, ..., L_{1k}$, then C can be solved without solving any of $L_{10}, ..., L_{1k}$ and according to

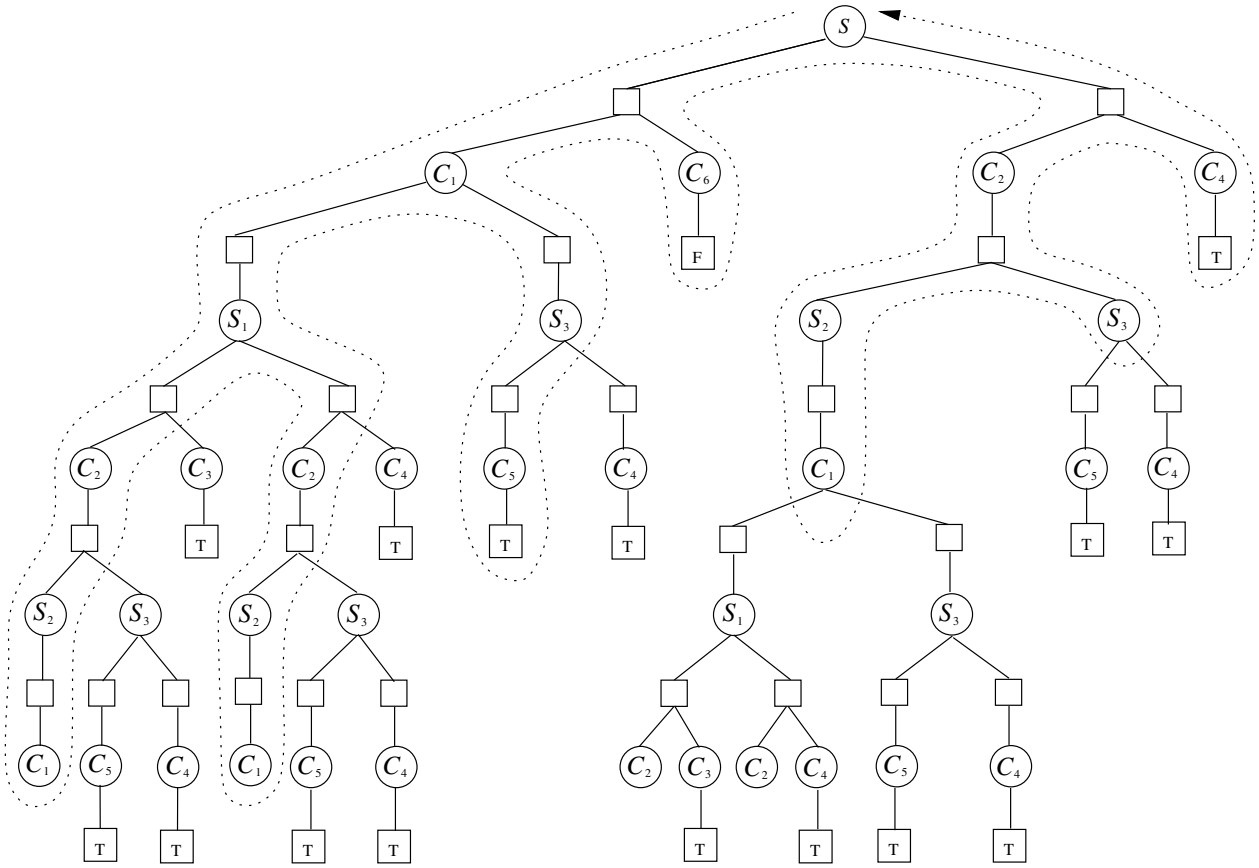

Figure 2: The negotiation search tree for the disclosure policies given in figure 1. S is the service the server may provide. Round nodes represent credential nodes while square nodes represent disjunctive nodes. The dotted line represents the search path taken by the SAs during negotiation using the brute-force backtracking strategy.

Theorem 1, R_1 should not be denied in the first place, leading to a contradiction. So E must contain some credentials in $L_{1\ t+1}, ..., L_{1k}$. Let L_{1f} be the first such credential to appear in E. Since L_{1f} is in p_1, when R_1 was denied, $\mathcal{G}$ does not contain any credentials in $L_{1\ t+1}, ..., L_{1k}$. We know that when R_2 arrives, $|\mathcal{G}|$ has not changed since R_1 was denied. Therefore L_{1f} was denied during the backtracking to L_{1t}, before R_2 arrives. According to Theorem 1, to solve L_{1f}, at least one of $L_{10}, \cdots, L_{1\ f-1}$ must be solved. Since E does not contain any of $L_{10}, ..., L_{1t}$, at least one of $L_{1\ t+1}, ..., L_{1\ f-1}$ has been solved before L_{1f} is solved, which is a contradiction with that L_{1f} is the first one in $L_{1\ t+1}, ..., L_{1k}$ to be solved.

The proof for (2) is similar. Let *period 1* cover the time from the receipt to the denial of R_1, and let *period 2* cover the time from the receipt to the denial of R_2. Suppose when R_2 is denied, $|\mathcal{G}| > k$. Then some credentials have been granted during period 2. Let C' be the first of such credentials and p_3 be the path from C to C'.

If p_3 does not contain anything in p_1 except C, then during period 1, C' must have been requested through a path from C to C' that looks exactly the same as p_3, because there cannot be any circular dependencies along this path. Since C' is granted in period 2, it must exist and have been denied in period 1. According to Theorem 1, at least one credential along $p_1 + p_3$ was granted before C' is granted. Because C' is the first credential granted in period 2 and the set of granted credentials remains unchanged between the time when R_1 is denied and R_2 is received, no credentials in p_1 have been granted before C' is granted. Therefore, at least one credential in p_3 was granted when R_2 is received, which means the search along p_3 will not reach C' at all in period 2, but stop and backtrack at the granted credential – a contradiction.

If p_3 contains credentials in p_1 other than C, they should be in $C_{1\ t+1}, \ldots, C_{1k}$ because otherwise there would be a circular dependency with credentials in p_2. Suppose L is the closest of these shared credentials to C' in p_3. Note that a node for C' is also a descendent of the L that appears in p_1, and C' must have been requested when that node was traversed because there is a path with no circular dependencies from the root to C' via that L. Since C' is the first credential granted in period 2, the credential exchange sequence for C' does not contain anything from $p_1 + p_3$. Then the earlier request for C' must have been granted. Again we have a contradiction. □

Based on Theorem 2, we developed PRUNES, a refined backtracking strategy that avoids unnecessary request resending and policy rescanning for a credential when no progress has been made in the negotiation since the last time that

credential was denied. At each party, each credential C is associated with a variable called *current_granted* which is the size of $\mathcal{G}$ when the last request for C was denied. There is also a global variable *total_granted* at each party, recording the current size of $\mathcal{G}$ in the negotiation. Whenever a new credential is granted, *total_granted* is increased by one. When an SA tries to solve a credential C that was denied before, it first checks whether C's *current_granted* equals *total_granted*. If it does not, the SA will send a message to the opposite side requesting C. Otherwise, according to Theorem 2, C cannot be solved in the current situation. So the SA will backtrack without sending a request for C. Figure 3 gives the pseudo code of three key functions of PRUNES.

The second part of Theorem 2 ensures that by pruning the search tree this way, we will not miss any chance of granting other credentials, compared with the brute-force backtracking strategy. Meanwhile, the search path followed by PRUNES is a subsequence of that followed by the brute-force one. Therefore given the same policies and sets of credentials, the two strategies always find the same solution. In another word, these two strategies are equivalent. Hence PRUNES is also complete.

We have not yet explained how the negotiation process leads to actual credential exchange. In our negotiation strategy, each SA also maintains a dependency graph. The nodes in the graph represent credentials that are granted during the negotiation. Besides indicating the granted credential, each grant message also includes a solved clause from the disclosure policy of that credential. Upon receiving such a grant message, an SA will update its dependency graph by adding one edge from each credential in the received clause to the granted credential. There is also a special node in the dependency graph which represents the empty formula ϵ. When a granted credential is unprotected, an edge from the ϵ node to that credential is added. The party sending this grant message will update its own dependency graph in the same way, so that both sides have the same dependency graph. Then when a negotiation succeeds, both sides will have consensus over the credential exchange sequence, which is defined by a topological sort of the nodes, starting from ϵ to the target service S. Then the actual credential exchange starts and trust will be established. We call this phase the *credential exchange phase*, while the first phase is the *negotiation phase*, when the two parties exchange request, grant and deny messages to find a credential exchange sequence for successful trust establishment.

Figure 4 gives an example to show how PRUNES works. It gives the message exchange sequence in the negotiation, along with the dependency graph after service S is granted. At time point 5, the server denies a request for S_2 because S_2 needs C_1 and there is already a pending request for C_1 (circular dependency). At time point 7, when the server receives a deny message for C_2, it records the current total number of granted credentials (0 here) for C_2 and backtracks to the second clause in the policy for S_1, where the first credential is again C_2. Since the total number of granted credentials has not changed since the last time C_2 was denied, the server will not request C_2 at this point. It denies S_1 instead. However, at time point 15, the total number of granted credentials has increased to 1. The server will request C_2 then.

The following two theorems show the efficiency of PRUNES.

THEOREM 3. *The worst case communication complexity (total number of messages, total size of messages, and number of rounds) of PRUNES is $O(n^2)$, where n is the total number of credentials requested during the negotiation.*

PROOF. As mentioned before, the communication complexity of a negotiation strategy has two aspects: the total number of messages and the total size of messages.

During the credential exchange phase in PRUNES, there are at most n messages and the size of each message depends on the size of the credential (which we assume is bounded by a constant). We will then focus our discussion on communication occurring during the negotiation phase. During that phase, there are three kinds of messages: request, deny, and grant.

Since each credential is granted at most once, the number of grant messages is no more than the number of credentials of the two parties. Each grant message contains a clause in the credential's policy. We assume that credential name lengths are bound by a constant. So the size of a grant message is no more than $O(n)$. Therefore the cost of grant messages is $O(n^2)$ in the worst case.

A credential C is requested only when (1) it has not been granted yet and (2) it was never requested before or its *current_granted* is smaller than *total_granted*. If the request for C is denied, its *current_granted* is set to be equal to *total_granted*, which makes *current_granted* monotonically increasing. Since *total_granted* is at most the total number of credentials owned by the two parties, a credential C is requested at most that many times. Therefore the total number of request messages is $O(n^2)$ in the worst case. Since each request message only contains a credential name, the total size of request messages is also $O(n^2)$ in the worst case.

A deny message always responds to a request message and it also only contains a credential name. So the total number and size of deny messages are both $O(n^2)$ in the worst case.

In all, the worst case communication cost of PRUNES is $O(n^2)$, in the sense of both the total number of messages and the total size of messages. The number of rounds is half the number of messages. □

THEOREM 4. *The computational complexity of PRUNES is $O(nm)$, where n is the total number of credentials and m is the total size of the policies of both parties.*

PROOF. During the negotiation phase, the local computational cost for an SA is solely the cost of scanning the disclosure policies. For each arriving request for credential C, in the worst case, C's disclosure policy is completely scanned once. As shown in the proof of Theorem 3, there are at most n requests for C, so C's policy is scanned at most n times. The computational cost in this phase is therefore at most $O(nm)$.

During the credential exchange phase, the computational cost is dominated by the cost of performing a topological sort on the dependency graph. The dependency graph (same at both sides) has at most n vertices and n^2 edges, so the computational cost in this phase is at most $O(n + n^2) = O(n^2)$.

Since $m \geq n$, the total computational complexity is $O(nm)$. □

```
PREPARE-REQUEST(C)
    if (C has been granted) return true;
    if (C has a pending request) return false;
    if (this is the first time C is requested or
      C.current_granted < total_granted)
        SEND-MESSAGE(request(C));
        return(HANDLE-INCOMING-MESSAGE());
    else return false;

HANDLE-INCOMING-MESSAGE()
    while (true)
        msg = RECEIVE-MESSAGE();
        case (msg)
        deny(C):
            C.current_granted ← total_granted;
            return false;
        grant(C, D):
            total_granted ← total_granted + 1;
            return true;
        request(C): TRY-SOLVE(C);

TRY-SOLVE(C)
    if (C does not exist)
        SEND-MESSAGE(deny(C));
        return;
    if (C is unprotected)
        SEND-MESSAGE(grant(C, ε));
        total_granted ← total_granted + 1;
        return;
    done ← false;
    while (done = false and there is an untried clause)
        take the next clause D;
        solvable ← true;
        while (solvable = true and there is an untried credential)
            take the next credential L in D;
            if (PREPARE-REQUEST(L) = false)
                solvable ← false;
        if (solvable = true) done ← true;
    if (done = true)
        SEND-MESSAGE(grant(C, D));
        total_granted ← total_granted + 1;
    else
        SEND-MESSAGE(deny(C));
```

Figure 3: The pseudo code for three key functions of PRUNES. The client starts the negotiation by calling PREPARE-REQUEST(S), where S is the service requested by the client. Upon receiving the request for S, the server calls TRY-SOLVE(S).

The above analysis is about worst cases and only gives a loose upper bound on the number of messages. Also n and m here are only the number of credentials and the size of policies relevant to the negotiation, i.e., credentials requested and policies looked up during the negotiation. We can further reduce the number of messages by differentiating between the deny message for a credential not owned by the side receiving the request for that credential, and the deny message for a credential that incurs a circular dependency. In the former case, no request will be sent again for that credential. Then for credentials requested by one side but not owned by the other, there will be only one request and one deny message.

4. RELATED WORK

The works of Winsborough et al. [10, 11] have been discussed in detail earlier in the paper. The only thing we need to mention here is that in our opinion, the parsimonious strategy proposed in [11] resembles breadth first search in the negotiation search tree defined in the paper. A counter request corresponds to a new level of the search tree. Their strategy will finish the negotiation in $O(n)$ rounds, proportional to the height of the search tree, while the breadth of the search tree at different levels (which is exponential in the number of credentials in the worst case) corresponds to the size of different counter request messages.

The Trust Establishment Project at IBM Haifa Research Lab has developed a system that automatically assigns roles to strangers based on digital credentials and a role-assignment policy [5]. In their system, it is assumed that credentials (or certificates) are freely available. A certificate collector will automatically try to collect a user's credentials based on the role-assignment policy. Similar works include KeyNote [2], PolicyMaker [3] and REFEREE [4]. Because these works do not consider the problem of sensitive credentials, their trust establishment mechanisms are uni-directional. However, their works can be extended to allow bi-directional trust establishment using our approach.

Much work has been done on policy definition mechanisms. [5] defined a certificate-based role-assignment policy language based on XML, which maps a set of credentials to a role. Another paper [8] discussed the idea of mapping users to roles with attributes from certificates using the Prolog programming language. REFEREE [4] provides an extensible environment where interpreters for policy language can be loaded dynamically as long as they conform to the calling conventions. A single high-level API is maintained for trust decision.

The concept of an AND/OR tree is discussed in detail in [9]. Generally, an AND/OR tree is a representation of a problem decomposition. There is usually a cost associated with each subproblem as well as each decomposition.

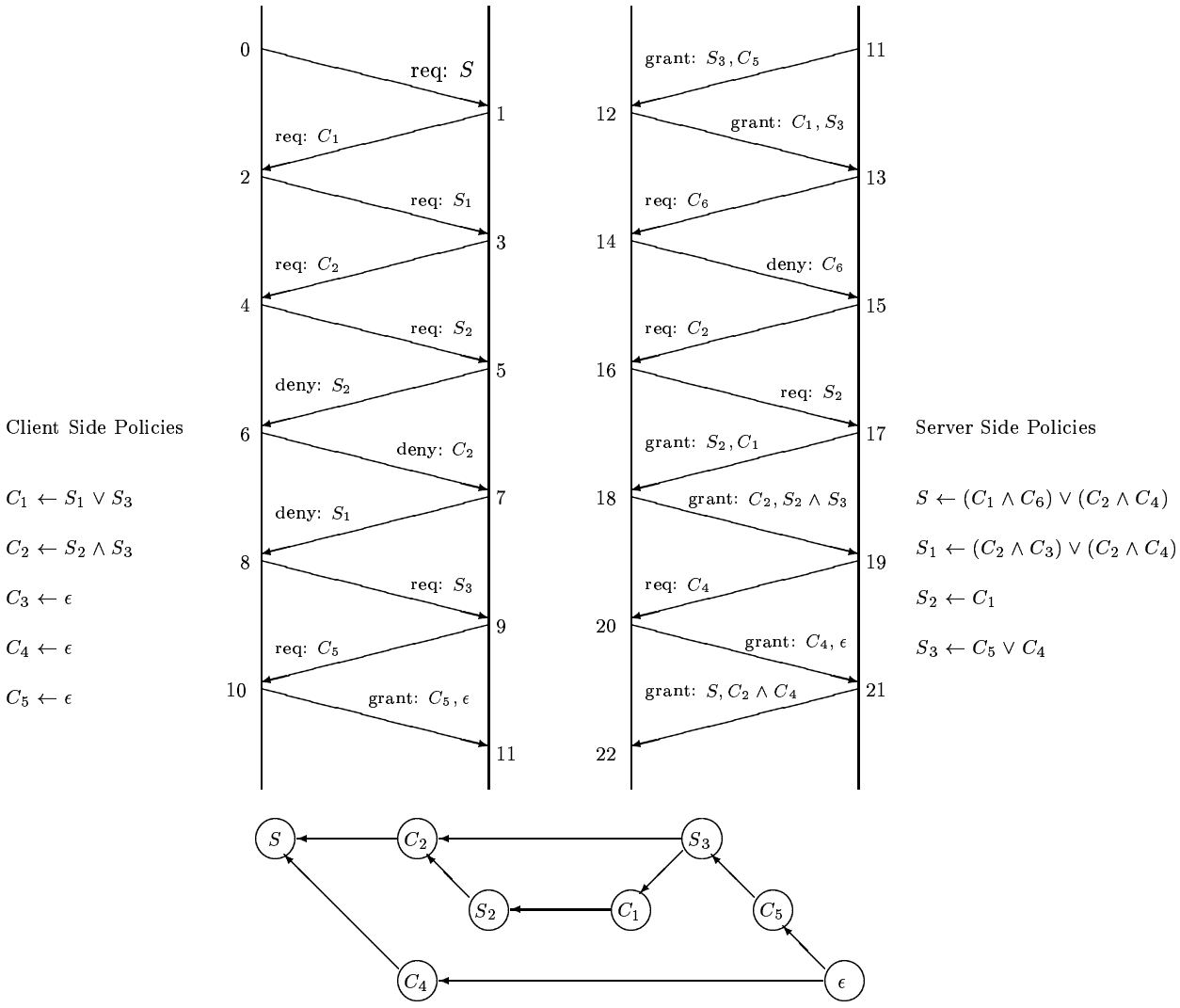

Figure 4: The message exchange sequence between the client and the server using PRUNES, along with the final dependency graph.

Given a monotonic heuristic cost function, a search algorithm called AO* can find the solution to a problem with the minimum cost. This algorithm could be used to minimize the number of credentials disclosed in a trust negotiation. However, in the worst case, AO* will have to traverse the whole AND/OR tree [6], implying exponential computational complexity.

Some previous work discussed the implementation of *negative credentials*. Sometimes a credential disclosure policy may state that this credential can be shown only if the other party does *not* have certain credentials. The policy language given in [5] can handle negative certificates by checking previously accumulated positive certificates and defining a way to evaluate a policy when some of the certificates in that policy are "undefined". We can replace our basic AND/OR tree by a more general model to handle negative credentials in a similar manner. [12] talked about several ways to implement the negative credentials. One of them is to transform a negative credential to a positive one. For example, we can have a credential stating that the owner does *not* have any criminal record. In that case, no adaptation needs to be made for our approach to handle negative credentials.

5. CONCLUSION AND FUTURE WORK

In this paper we presented the Prudent Negotiation Strategy (PRUNES), a complete automated trust negotiation strategy based on backtracking. PRUNES guarantees to find a successful negotiation whenever the credential policies of the service requester and provider allow. On the other hand, PRUNES ensures that no irrelevant credentials are disclosed in the resulting negotiation. We also proved the worst-case communication cost and computational complexity are $O(n^2)$ and $O(nm)$ respectively in a negotiation, where n is the total number of credentials requested and m is the total size of policies looked up during that negotiation.

There are many possible directions for extension of this work. In this paper, we have assumed all the credentials are of equal importance. In many situations, however, this assumption is not true. For example, one's credit card num-

ber or social security number is often much more sensitive than her home phone number. It is desirable to establish trust by exchanging the least sensitive credentials possible. Therefore, besides completeness and minimal credential disclosure, minimum total sensitivity of disclosed credentials might also be a desired feature of a negotiation strategy. We have suggested such an approach based on AO* search, but perhaps a cheaper alternative exists in some situations.

Another issue for trust negotiation is that on some occasions, the credential policy itself is also sensitive information. In this paper, whenever a credential request comes, we form a counter request according to the credential's policy and send it back. When the policy is also sensitive, we will have a disclosure policy for that policy too. Each credential will be associated with a *policy chain* instead of one single policy. How to design a negotiation strategy that deals with policy chains is still a challenging problem.

Finally, there should be protocols designed to allow negotiation between two SAs that are not following the same strategy. For example, the service provider is very eager to reach a deal, while the customer is super-cautious about releasing personal information. Negotiation under such circumstances should be faster than that discussed in this paper, yet without risking unnecessary credential disclosure from the more cautious side.

6. ACKNOWLEDGEMENTS

This research was supported by DARPA and AFRL under contract F30602-98-C-0222. Many thanks to Kent Seamons, Vicki Jones and Will Winsborough for helpful conversations and stimulating interaction. We would also like to thank Mattox Beckman for interesting discussions and comments, and the referees for their helpful suggestions.

7. REFERENCES

[1] E. Bina, V. Jones, R. McCool and M. Winslett, Secure Access to Data Over the Internet, in *Proceedings of the 3rd 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, http://www.cis.upenn.edu/~angelos/keynote.html.

[3] M. Blaze, J. Feigenbaum, and J. Lacy, Decentralized Trust Management, in *IEEE Symposium on Security and Privacy*, Oakland, CA, pp. 164-173, May 1996.

[4] Y. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick and M. Strauss, REFEREE: Trust Management for Web Applications, in *World Wide Web Journal*, 2, pp. 127-139, 1997.

[5] 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, CA, May 2000.

[6] L. Kanal and V. Kumar, *Search in Artificial Intelligence*, Springer-Verlag, 1988.

[7] B. Schneier, *Applied Cryptography*, John Wiley and Sons, Inc., second edition, 1996.

[8] K. Seamons, W. Winsborough, and M. Winslett, Internet Credential Acceptance Policies, in *Proceedings of the Workshop on Logic Programming for Internet Applications*, Leuven, Belgium, July 1997.

[9] R. Shinghal, *Formal Concepts in Artificial Intelligence*, Chapman & Hall Computing, 1992.

[10] W. Winsborough, K. Seamons and V. Jones, Negotiating Disclosure of Sensitive Credentials, in *Second Conference on Security in Communication Networks*, Amalfi, Italy, September 1999.

[11] W. Winsborough, K. Seamons and V. Jones, Automated Trust Negotiation, submitted for journal publication, April 2000, http://www.csc.ncsu.edu/faculty/vej/atn.ps.

[12] M. Winslett, N. Ching, V. Jones, and I. Slepchin, Using Digital Credentials on the World-Wide Web, in *Journal of Computer Security*, 1997.