Interoperable Strategies in Automated Trust Negotiation

Ting Yu
University of Illinois,
Urbana-Champaign

tingyu@cs.uiuc.edu

Marianne Winslett
University of Illinois,
Urbana-Champaign

winslett@cs.uiuc.edu

Kent E. Seamons
Brigham Young University

seamons@cs.byu.edu

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, e.g., 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, 11, 14]. Johnson 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 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 [13] 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. In work not described in this paper, we have extended SSL to serve as the substrate for private, secure trust negotiation.

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 [14]. Yu et al. [15] introduced a new strategy that would succeed whenever success was possible and had certain efficiency guarantees. In [12], 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

We establish trust 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. 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.

It is clear that there are endless possible variations in how to negotiate trust. In this paper we 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 language with a well-defined semantics. XML or logic programming languages with appropriate semantics may be suitable in practice [8, 1]. For convenience, we assume that the language allows us to describe the meaning of a set X of statements as the set of all models that satisfy X, in the usual logic sense. We say that X *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 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.

In this paper, we will treat credentials and services as propositional symbols. Each of these resources has one access control policy, of the form $C \leftarrow F_C(C_1, \ldots, C_k)$, where $F_C(C_1, \ldots, C_k)$ is a Boolean expression involving only credentials $C_1, \ldots, C_k$ 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 $G = (C_1, \ldots, C_n)$ of disclosures of protected resources, if each C_i is unlocked at the time it is disclosed, then we say G is a *safe disclosure sequence*. The goal of trust negotiation is to find a safe disclosure sequence where $C_n = R$, the resource to which access was originally requested. When this happens, we say that trust negotiation succeeds. If $C_i = C_j$ and $1 \le i < j \le n$, then we say G is *redundant*. Language monotonicity allows us to 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 (section 1). A more complex example can be found in the full version of this paper [16]. Recall 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.

5. TRUSTBUILDER PROTOCOL AND STRATEGY FAMILIES

Previous work has not explicitly proposed any trust negotiation protocols, instead defining protocols implicitly by the way each strategy works. This is one reason 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 $\{R_1, \ldots, R_k\}$ 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 call it a *failure message*. 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 $C \leftarrow false$, 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 (figure 3).

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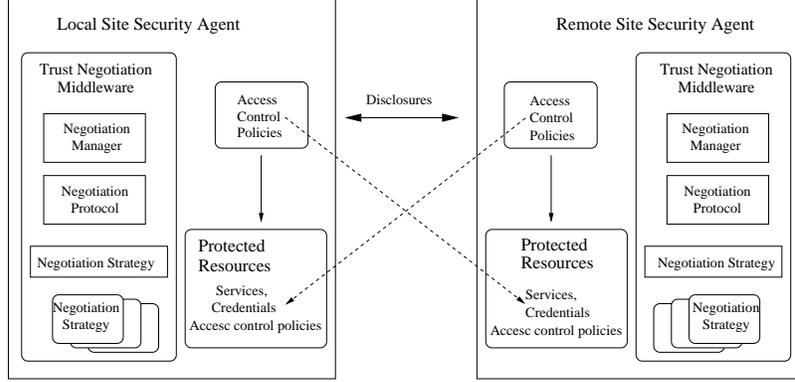
Figure 1: An architecture for automated trust negotiation. A security agent that manages local protected resources and their associated access control policies represents each negotiation participant. A access control policy specifies what resources the other party needs to disclose in order to gain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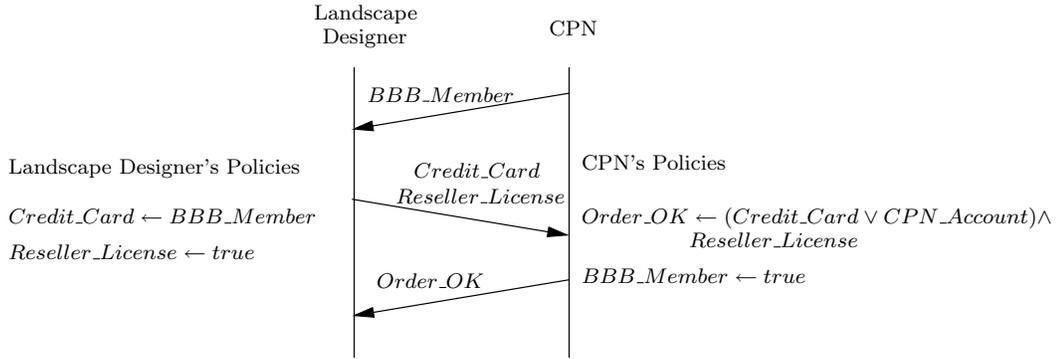

Figure 2: An example of access control policies and a safe disclosure sequence which establishes trust between the server and the client.

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,* $G = (m_1, \ldots, m_k)$ *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.*

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 strategy 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, the SA may adopt a less cautious strategy. Without the

```
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 = 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 ∅ 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.

closure property, a family may be too small 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.

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 not 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.

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_3 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 are included in the full version of this paper [16].

THEOREM 6.1. *Given a non-redundant safe disclosure sequence $G = (C_1, \ldots, C_n = R)$, 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 $\{C_1, \ldots, C_n\}$.*

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.* □

From theorems 6.1 and 6.2, we have:

COROLLARY 6.1. *Given a safe disclosure sequence $G = (C_1, \ldots, C_n = R)$, there is a full non-redundant disclosure tree T such that:*
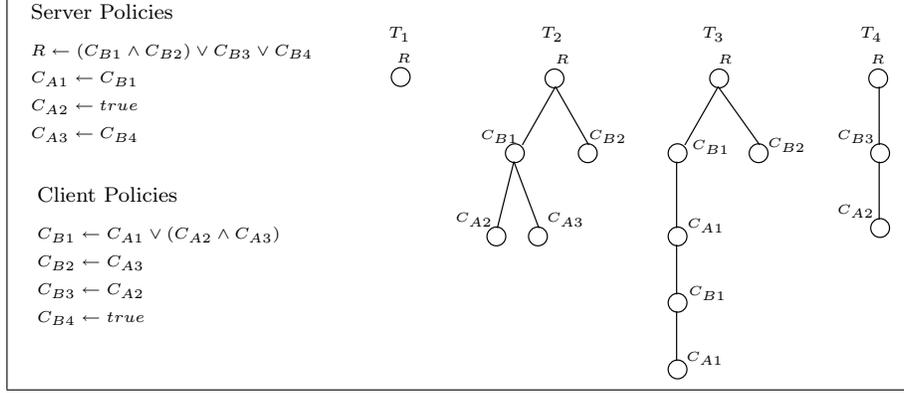
Figure 4: Example disclosure trees for a set of policies

1. T's credential nodes are a subset of $\{C_1, \ldots, C_n\}$.

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 credentials S_c, the* reduction *of T by S_c, 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,*

$$reduction(S_t, S_c) = \{reduction(T, S_c) \mid T \in S_t\}.$$

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, 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 $reduction(T_i, S) = T$.*

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,

$$expansion(S_t, S_p) = \bigcup_{T \in S_t} expansion(T, S_p).$$

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 $prune_{denial}(S_t, S_{dp})$, is the set*

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

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 irrelevant.

DEFINITION 6.5. *Given a set S_t of disclosure trees, the* redundancy pruning *of S_t, denoted $prune_{redundant}(S_t)$, is the set*

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

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 evolution(T, S), is*
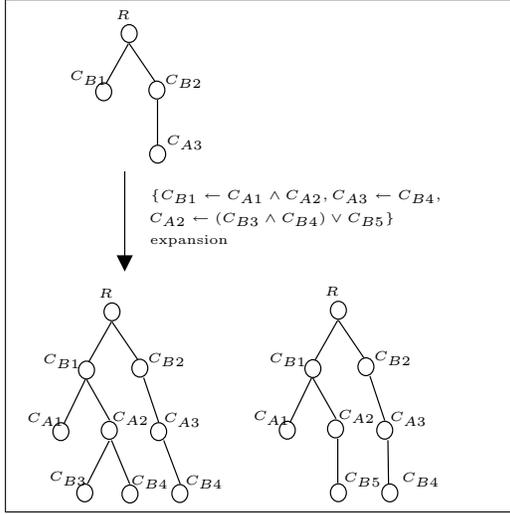
$$prune_{redundant}(prune_{denial}(reduction(\\ expansion(T, S_p), S_c), S_{dp}).$$

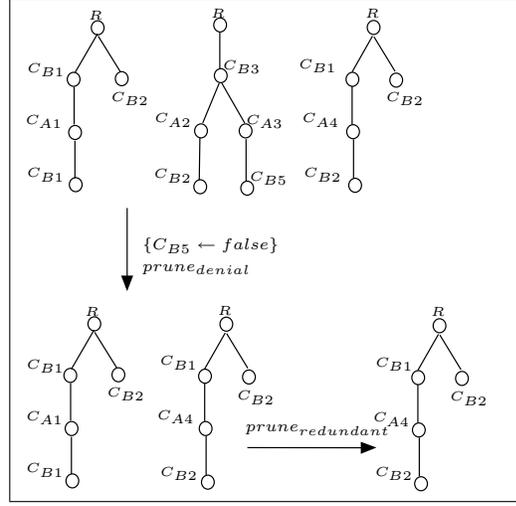Given a set S_t of disclosure trees,

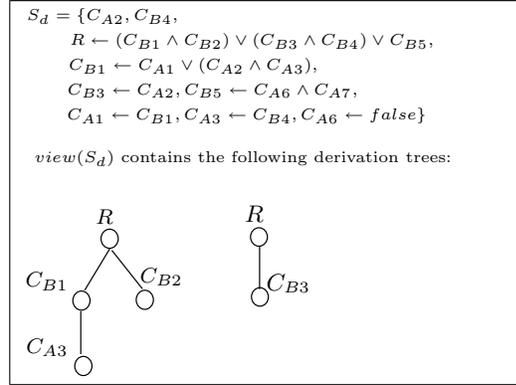$$evolution(S_t, S) = \bigcup_{T \in S_t} evolution(T, S).$$
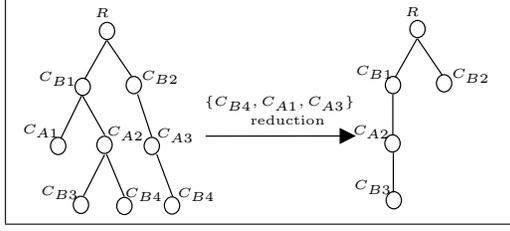
(a) Example of a disclosure tree reduction

(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

As a special case, when T is the disclosure tree containing only a root node R, then we say evolution(T, S) is the view *of S, denoted 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 $view(S \cup L)$ contains all the relevant disclosure trees that 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 current set of 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 $view(S)$, and the redundant trees can be ignored.

To ensure negotiations succeed whenever possible, a negotiation strategy cannot overlook any possible full disclosure trees. 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 $view(S_d)$ 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 shows that to let 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 $\mathcal{P}_A$, denoted $evolvable(T, \mathcal{P}_A)$, 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 $evolvable(T, \mathcal{P}_A) \neq \emptyset$, 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 $evolvable(T, \mathcal{P}_A)$ 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.*

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*

$$\forall m \in f_2(G, L, R)\ \exists m' \in f_1(G, L, R)\ m' \subseteq m$$

then we say f_1 is at least as cautious as *f_2, denoted $f_1 \preceq f_2$ 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, 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 DTS FAMILY

We present the *disclosure tree strategy* (DTS), then prove that DTS generates a closed family. Throughout this section, we assume that $G = (m_1, \ldots, m_k)$ 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 *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 credentials in S_d unlock R;*
   - *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) tells when DTS will terminate the negotiation. Condition 2) guarantees that the other party will have an evolvable tree, so the other party can send a message that further evolves a 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 $f' \neq f_2$.*

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.* □

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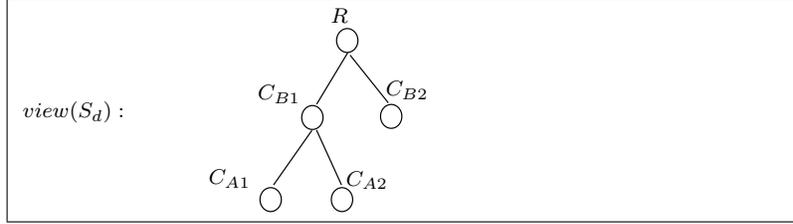
Figure 6: $view(S_d)$ **where** $S_d = \{R \leftarrow C_{B1} \wedge C_{B2}, C_{B1} \leftarrow C_{A1} \wedge C_{A2}\}$

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.

TrustBuilder-Simple (figure 7(a)) 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 fails.

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.

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.* □

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.* □

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

This paper focused on guaranteeing interoperability between different strategies. We first proposed a very simple trust negotiation protocol for the TrustBuilder trust negotiation architecture. Then we studied strategies that adhere to this protocol. We introduced 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 introduced the concept of disclosure trees and identified the natural mapping between full disclosure trees and safe credential disclosure sequences. We then proposed the disclosure tree strategy (DTS), and proved that all the strategies that are no more cautious than DTS form a closed strategy family. Finally we gave 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.

9. ACKNOWLEDGEMENTS

10. 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.
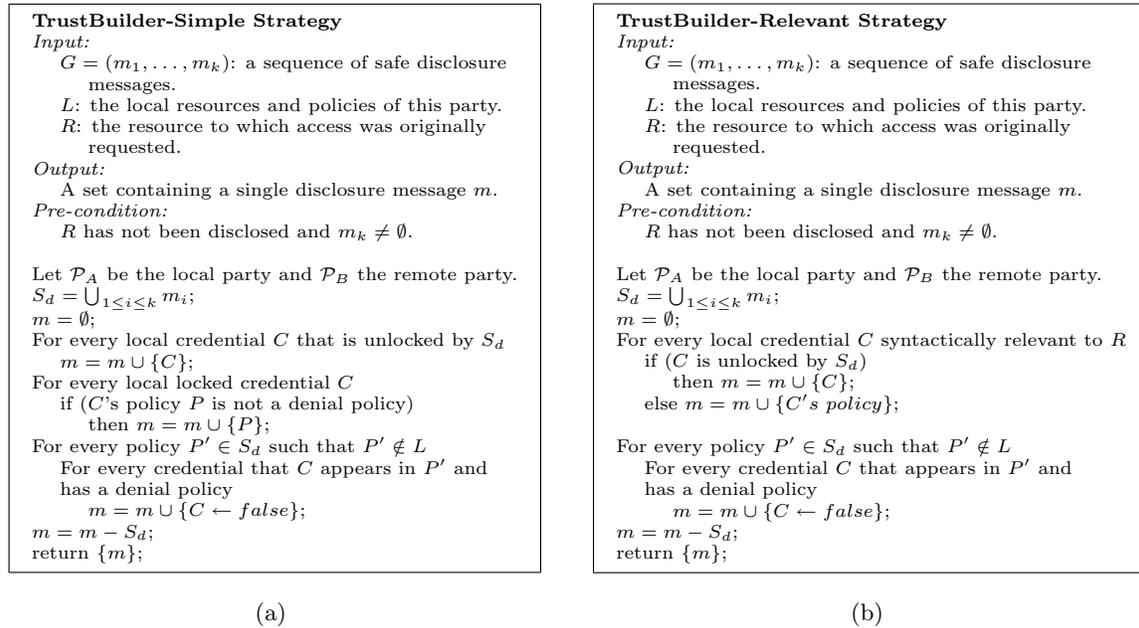
```
TrustBuilder-Simple Strategy
Input:
    G = (m_1, ..., m_k): a sequence of safe disclosure
        messages.
    L: the local resources and policies of this party.
    R: the resource to which access was originally
        requested.
Output:
    A set containing a single disclosure message m.
Pre-condition:
    R has not been disclosed and m_k ≠ ∅.

Let P_A be the local party and P_B the remote party.
S_d = ⋃_{1≤i≤k} m_i;
m = ∅;
For every local credential C that is unlocked by S_d
    m = m ∪ {C};
For every local locked credential C
    if (C's policy P is not a denial policy)
        then m = m ∪ {P};
For every policy P' ∈ S_d such that P' ∉ L
    For every credential that C appears in P' and
    has a denial policy
        m = m ∪ {C ← false};
m = m − S_d;
return {m};
```

```
TrustBuilder-Relevant Strategy
Input:
    G = (m_1, ..., 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 ≠ ∅.

Let P_A be the local party and P_B the remote party.
S_d = ⋃_{1≤i≤k} m_i;
m = ∅;
For every local credential C syntactically relevant to R
    if (C is unlocked by S_d)
        then m = m ∪ {C};
    else m = m ∪ {C's policy};

For every policy P' ∈ S_d such that P' ∉ L
    For every credential C that appears in P' and
    has a denial policy
        m = m ∪ {C ← false};
m = m − S_d;
return {m};
```

(a) (b)

Figure 7: Pseudocode for two strategies in the DTS family

[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] 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.

[12] 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.

[13] W3C, http://www.w3.org/TR/WD-P3P/Overview.html. *Platform for Privacy Preferences (P3P) Specification*.

[14] W. Winsborough, K. Seamons, and V. Jones. Automated Trust Negotiation. In *DARPA Information Survivability Conference and Exposition*, Hilton Head, SC, January 2000.

[15] 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.

[16] T. Yu, M. Winslett, and K. Seamons. Interoperable Strategies in Automated Trust Negotiation. In *http://drl.cs.uiuc.edu/pubs/ccs2001-long.ps*.

[17] P. Zimmerman. *PGP User's Guide*. MIT Press, 1994.