Automated Trust Negotiation

William H. Winsborough
IBM Transarc Lab
winsboro@us.ibm.com

Kent E. Seamons
IBM Transarc Lab
seamons@us.ibm.com

Vicki E. Jones
North Carolina State University
jones@csc.ncsu.edu

Abstract

Distributed software subjects face the problem of determining one another's trustworthiness. The problem considered is managing the exchange of credentials between strangers for the purpose of property-based authentication and authorization when credentials are sensitive. An architecture for trust negotiation between client and server is presented. The notion of a trust negotiation strategy is introduced and examined with respect to an abstract model of trust negotiation. Two strategies with very different properties are defined and analyzed. A language of credential expressions is presented, with two example negotiations illustrating the two negotiation strategies. Ongoing work on policies governing credential disclosure and trust negotiation is summarized.

Keywords: *Trust Establishment, Trust Management, Agent Negotiation, Access Control, Digital Credentials, Certificates, Privacy.*

1. Introduction

Distributed software subjects face the problem of determining one another's trustworthiness. Current mainstream approaches to establishing trust presume that communicating subjects are already familiar with one another. There are essentially two approaches based on this assumption. The first is identity-based: identifying a subject is often a sufficient basis for doing business. The second is capability-based: subjects obtain capabilities that are specific to the resources they wish to use. Both approaches require that familiarity be established out of band. Identity- and capability-based approaches are both unable to establish trust between complete strangers. Other solutions are needed in open systems, such as the web, where the assumption of familiarity is invalid.

Property-based *digital credentials* [1] (or simply *credentials*) are the on-line analogues of paper credentials that people carry in their wallets. They present a promising approach to trust establishment in open systems. Credentials, which generalize the notion of attribute certificates[19], can authenticate not just the subject's identity, but arbitrary properties of a subject and its relationships with other subjects. To support service authorization, a client can attach appropriate credentials to service requests.

Trust establishment between strangers is particularly important in the context of e-business. Credential exchange between strangers promises to enable software agents to establish trust automatically with potential business partners. For instance, a software agent might be charged with finding new candidate suppliers of commodity goods and services. Even when an automatically generated list of such candidates eventually would be culled by a human, information such as requirements and availability could be sensitive, requiring trust establishment as part of the automated process of identifying candidates.

This paper presents automated trust establishment between strangers through credential exchange when credentials are themselves potentially sensitive. A *sensitive credential* contains private information. For instance, access to a credential containing medical information could be restricted to primary care physicians and HMO staff. Access to a credit card credential could be limited to businesses that are authorized to accept a VISA card and that adhere to guidelines for securing credit card numbers. Prior trust establishment systems based on credential exchange have addressed credential sensitivity only manually, requiring a user at the client to decide which credentials to submit to each new service. In addition to requiring human intervention, this approach provides to the human making the trust decision no assistance in evaluating the trustworthiness of the server.

This paper presents an architecture for client-server applications in which client and server each establishes a *credential access policy* (CAP) for each of its credentials. A credential is disclosed only when its CAP is satisfied by credentials obtained from the opposing software agent. When an agent needs additional credentials, it can request them. Credentials flow between the client and server through a sequence of alternating credential requests and disclosures, which we call a *trust negotiation*.

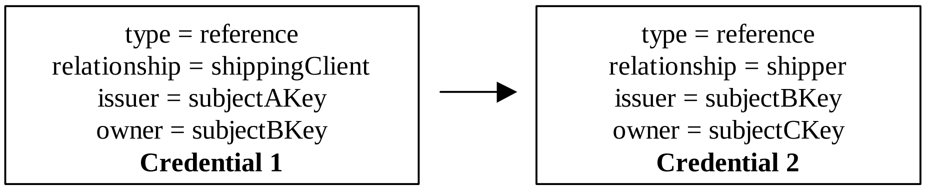


Figure 1. Two credentials forming a chain. Credential 2 was issued by subject B, the owner of Credential 1. In Credential 1, subject A asserts that subject B is a consumer of shipping services. In Credential 2, subject B asserts that subject C is a shipper. If we trust subject A's judgment that subject B is a consumer of shipping, presumably subject B is in a position to know that subject C is a shipper. Additional credentials owned by subject B can be used to engender trust that subject B is a reliable authority on the asserted attributes of subject C.

A negotiation strategy determines characteristics of a negotiation such as which credentials are requested and when the negotiation is halted. We discuss two negotiation strategies with very different properties.

Participants using the first strategy turn over all their credentials as soon as their CAPs are satisfied, without waiting for the credentials to be requested. For this reason we call it an *eager* strategy. It is simple and efficient, and leads to successful negotiation whenever possible. However, it discloses more credentials than necessary to achieve most trust requirements. Participants using the second strategy exchange credential requests that focus the credential exchange, achieving a kind of local minimality of disclosures. For this reason, we call it a *parsimonious* strategy. It is also reasonably efficient, and succeeds whenever possible. However, it has the drawback that the credential requests can disclose sensitive information about the subjects' credentials and properties. This paper discusses these tradeoffs and techniques for managing them.

Section 2 introduces credentials and explains how they can be used to establish trust between strangers. Section 3 introduces credential sensitivity, clarifying the present technical contribution. Section 4 presents a trust negotiation architecture and an abstract model of trust negotiation that is used in Section 5, where eager and parsimonious negotiation strategies are discussed. Section 6 presents two expression languages that can be used within the trust negotiation architecture to express CAPs and credential requests. Section 7 presents two examples of trust negotiation, illustrating the eager and parsimonious negotiation strategies, respectively. Section 8 discusses related work. Section 9 draws conclusions and presents future research directions.

2. Credential-based Trust

A credential is a digitally signed assertion by the *credential issuer* about the *credential owner*. Credentials can be made unforgeable and verifiable by using modern encryption technology: a credential is signed using the issuer's private key and verified using the issuer's public key [12]. A credential aggregates one or more *attributes* of the owner, each consisting of an attribute name/value pair and describing some property 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 possession of that private key to establish ownership of the credential. The owner can also use the private key to sign another credential, owned by a third subject. In this way, *credential chains* can be created, with the owner of one credential being the issuer of the next credential in the chain.

Credential chains can be submitted to trace a web of trust from a known subject, the issuer of the first credential in the chain (e.g., subject A in Figure 1), to the *submitting subject*, in which trust is needed. The submitting subject is the owner of the last credential in the chain (e.g., subject C) and can demonstrate ownership of that credential, as outlined above. *Supporting* credentials are owned by subjects with whom the submitting subject has a direct or indirect relationship, and, although they are not owned by the submitting subject, the submitting entity does collect, keep, and submit copies of them. Each supporting credential contains the public key whose private-key mate signed the next credential in the chain, enabling reliable verification that the attribute claims made in that next credential were made by the owner of the supporting credential.

The submitted credentials attempt to demonstrate a (possibly indirect) relationship between the submitting subject and the known subject that issued the first credential in the chain. The nature of that relationship can be inferred by inspecting the attributes of the credentials in the chain. Multiple chains can be submitted to establish a higher degree of trust or to demonstrate additional properties of the submitting subject and its relationships with known subjects.

A *credential expression*, y, is a logical expression over credentials with constraints on their attributes. A credential expression serves to denote the combinations of credentials, *C*, that satisfy it. We call those combinations the *solutions* of the expression. For the purpose of trust negotiation, credential expressions can be used to convey requests for credentials between client and server. In this context, credential expressions denote chains of credentials that end with credentials owned by the submitting subject. A credential expression can also be used as a *policy* governing access to a resource. Access to the resource is granted to a subject when a solution is presented that consists of one or more chains ending in credentials owned by the subject. The resource is *unlocked* by the solution.

A policy is *mobile* if it is sent from one subject to another as part of automatic or semiautomatic trust establishment. Mobile policies are used in prior systems to express requirements a client must meet to obtain service. When insufficient credentials accompany a service request, the server returns the *service-governing policy* (SGP). Communicated in this way, the SGP acts as a request for the credentials needed to unlock the resource. Such mobile policies enable clients to select a minimal set of credentials whose submission will authorize the desired service. The client can then issue a second request for service with those credentials attached, and upon verifying the credentials, the server provides the desired service. Policy mobility has two significant advantages. First, it offloads from the server to the client the work of searching the client's credentials. Second, it enables trust to be established in the client without the client revealing irrelevant credentials.

3. When Credentials are Sensitive

A client wishing to do business with a new service may be unwilling to disclose sensitive credentials until some degree of trust has been established in that service. Current credential systems do not address credential sensitivity. The decision to disclose a sensitive credential to a new service is left up to a user at the client. More specifically, client-credential submission policies specify which credentials can be submitted with any request to a specified class of service and which credentials require explicit authorization before they are submitted. This mechanism requires a user be available to make trust decisions when new service classes are contacted. It does not address how the user decides to trust a service.

In [16], Winslett et al. recognize the potential to use server credentials to establish client trust, though they focused mainly on server trust in clients. They present detailed mechanisms for clients to submit credentials to servers. Each service provider establishes a policy governing access to its resources. Upon receiving a request with insufficient credentials attached—often the case on first access—a security agent representing the server sends the client an explanation of the relevant portions of the policy. The client's agent analyzes the server's policy to determine which credentials are needed to support the service request. It makes a decision on which credentials to send to the server (based on a pre-defined client submission policy and interaction with the user). The client agent then attaches the selected credentials to subsequent requests for the service.

Winslett et al. note the relevance of such machinery for the reverse scenario, in which servers encourage clients by presenting their own credentials, and a client may request credentials from a server. Such a reversal provides a good basis for clients establishing the trust in servers required before disclosing sensitive credentials. However, [16] is unclear about the details of how this kind of trust can be established. The current paper describes recent work on automating the establishment of trust between strangers through *incremental* exchange of sensitive credentials.

One straightforward approach that unfortunately does not work is as follows. The client establishes a policy identifying credentials required from the server prior to the client disclosing any credentials. The client sends this policy to the server as a counter request whenever it receives from the server a credential request, such as a SGP. The problem is that it would then be useless for the server to request client credentials before disclosing its own credentials, as doing so would introduce a cyclic dependence and deadlock. This simplistic approach fails because it governs all client credentials with the same policy, so each request by the server for client credentials leads to an identical counter request from the client.

4. Negotiation Architecture and Model

Section 4.1 presents a high-level architecture that supports fully automated trust negotiation between client and server. Section 4.2 then introduces an abstract model of trust negotiation. The model is used in Section 5 to formalize and analyze two negotiation strategies.

In both architecture and model, each credential is protected by a CAP that controls the credential's disclosure based on credentials presented by the other negotiation participant. Throughout, credentials are disclosed only in observance of these CAPs. Credential requests can also be exchanged to guide the credential exchange.

4.1. Trust Negotiation Architecture

Each of our negotiation participants is represented in trust negotiations by a *security agent* (SA), as in the simple negotiations of Ching et al. [5] and Winslett et al.

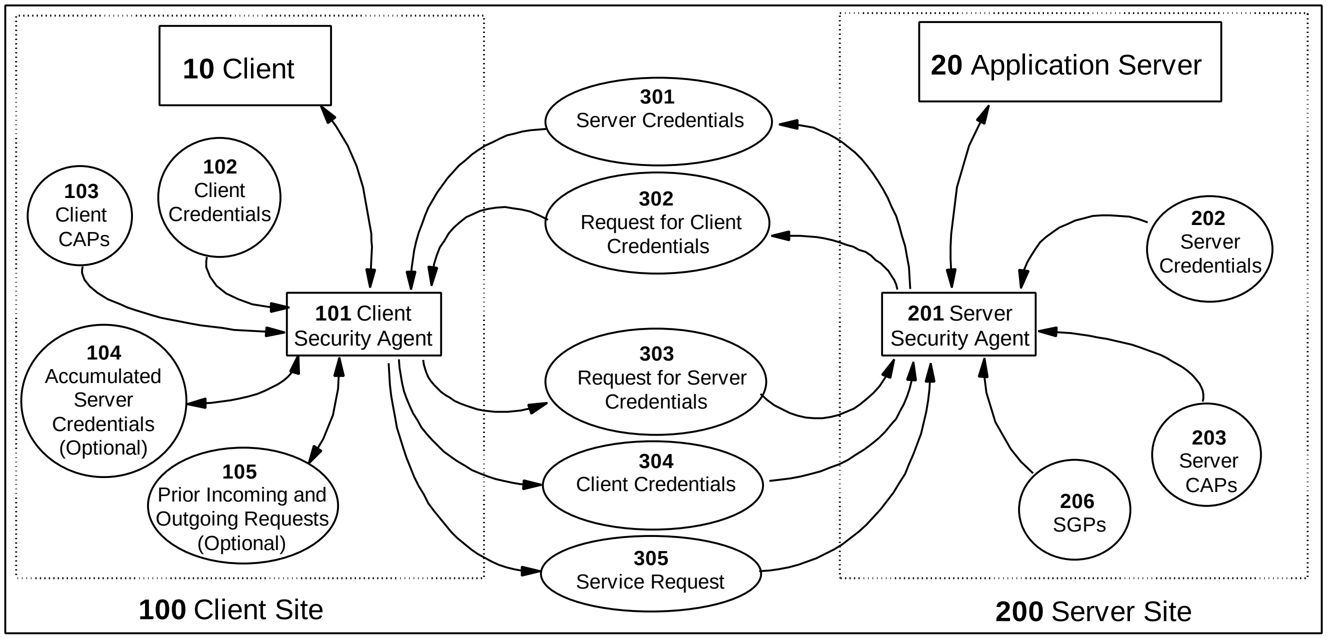

10 Client
20 Application Server
301 Server Credentials
302 Request for Client Credentials
103 Client CAPs
102 Client Credentials
202 Server Credentials
101 Client Security Agent
201 Server Security Agent
104 Accumulated Server Credentials (Optional)
303 Request for Server Credentials
304 Client Credentials
203 Server CAPs
105 Prior Incoming and Outgoing Requests (Optional)
206 SGPs
305 Service Request
100 Client Site
200 Server Site

Figure 2. The role of security agents in trust negotiation.

[16]. The role of the SA is illustrated in Figure 2, which depicts the client security agent (101) and the server security agent (201) and several contextual factors that each SA considers during negotiation. The client SA manages the disclosure of client credentials (102) and the server SA manages disclosure of server credentials (202). Like any protected resource, each credential is governed by an access policy (103, 203) that has the same form as a SGP. The CAP identifies credentials from the other negotiation participant that would unlock disclosure of the local credential to that subject.

The client (10) initiates the trust negotiation by making a service request. The client SA intercepts the request and relays it to the server SA. The service (20) is accessible only via the server SA. Upon receiving a request for service (305), the server SA makes an authorization decision based on the appropriate SGP (206). When the client SA is familiar with the SGP, it can attach appropriate credentials (304) to the service request so that the service will be authorized. The server SA determines whether the credentials that arrive with the service request satisfy the SGP. If the policy is satisfied, the trust negotiation has completed successfully; the service is authorized and the request is forwarded to the application server, which provides the service to the client.

Initially, the client is unfamiliar with the SGP, so attaching satisfactory credentials to an initial service request is impractical. A trust negotiation strategy can overcome this problem by using mobile policies. When a server SA receives a request for service without sufficient credentials attached to satisfy the SGP, it sends the SGP to the client SA as a request for client credentials (302). The client SA can then select a combination of credentials that satisfies the SGP, and can attach those credentials (304) to a repetition of the original service request (305).

An important issue in this scenario not addressed in previous trust systems is how to enable the client SA to make independent trust decisions about which credentials to provide to an unfamiliar server. Our SAs use CAPs (103, 203) when selecting credentials to disclose. If the client SA cannot satisfy the SGP by using credentials whose CAPs are unprotected, it can, as the *negotiation instigator*, introduce further stages to the trust negotiation by requesting server credentials (303). These stages seek to build mutual trust through credential exchange, eventually to unlock client credentials that satisfy the SGP. Client credentials are unlocked by incoming server credentials (301); however, as an optimization, the client may also cache and use for this purpose server credentials it received in prior stages (104). (The abstract model of trust negotiation introduced in Section 4.2 does not capture this optimization.)

In each negotiation stage, the active subject responds to an incoming request for credentials either by providing credentials, by formulating a counter request for credentials (302, 303), or both. In some strategies, the client SA can also repeat one of its previous request (105) for credentials that has not yet been satisfied. By exchanging credentials and requests for credentials, the two subjects endeavor to establish trust required to authorize service. Eventually, either the negotiation succeeds or the client SA must abandon the attempt. The negotiation succeeds when the client SA satisfies the original SGP by disclosing sufficient unlocked credentials. At the same time, the client SA repeats the original service request (305), this time with sufficient credentials attached (304) to authorize service.

4.2. Trust Negotiation Model

In this section we introduce the abstract model used in Section 5 to define and analyze two negotiation strategies. That abstract model formalizes a trust negotiation as a sequence of credential disclosures that alternate between the two participants, optionally augmented by a sequence of credential requests that serve to guide the disclosures.

The participants in a trust negotiation are the client and server. Each owns a set of credentials, which we denote by ClientCreds and ServerCreds, respectively. Access to each credential c in ClientCreds or ServerCreds is governed by a policy, denoted $\mathsf{gov}_{\mathsf{client}}(c)$ or $\mathsf{gov}_{\mathsf{server}}(c)$, respectively. If a credential expression, ψ, is satisfied by a set of credentials C, we write $\mathbf{sat}(C, \psi)$. We write $\psi \equiv \psi'$ if for all credential sets C, $\mathbf{sat}(C, \psi)$ *iff* $\mathbf{sat}(C, \psi')$. If $C \subseteq$ ClientCreds and $c \in$ ServerCreds such that $\mathbf{sat}(C, \mathsf{gov}_{\mathsf{server}}(c))$, or if $C \subseteq$ ServerCreds and $c \in$ ClientCreds such that $\mathbf{sat}(C, \mathsf{gov}_{\mathsf{client}}(c))$, we write $\mathbf{unlocked}(c, C)$. If $\mathbf{unlocked}(c, \emptyset)$, c is *unprotected*. Lifting to sets of credentials, C', we write $\mathbf{unlocked}(C', C)$ if $\mathbf{unlocked}(c, C)$ holds for each $c \in C'$.

A *trust negotiation* is formalized by a sequence of credential disclosures, $\{C_i\}_{i\in[0,2n+1]} = C_0, C_1, \ldots, C_{2n+1}$, for some natural number n, that alternates between the two subjects, *i.e.*, $C_{2i} \subseteq$ ClientCreds and $C_{2i+1} \subseteq$ ServerCreds for all i, $0 \leq i \leq n$. Each disclosure corresponds to a message. The number of disclosures is even, modeling a sequence of roundtrips.

The credentials in each disclosure are required to be unlocked by credentials from the other negotiation participant in the previous disclosure, which means that the first disclosure consists entirely of credentials that are unprotected. That is, we have $\mathbf{unlocked}(C_0, \emptyset)$ and $\mathbf{unlocked}(C_{i+1}, C_i)$ for all $0 \leq i \leq 2n$. Any disclosure can be empty, provided the subsequent disclosure consists of unprotected credentials. (This possibility is exercised, for instance, when modeling client/server interactions where the first disclosure is done by the server or the last disclosure is done by the client.)

Both client and server may set *trust requirements* that a trust negotiation may or may not succeed in establishing. Trust requirements are formalized by credential expressions. The trust requirement of primary concern in this paper is the server's policy governing access to a service: the SGP. However, a client might also set a trust requirement that it enforces before doing business with a server.

A trust negotiation is *successful* with respect to a server-set trust requirement, ψ, if the client's final disclosure satisfies ψ, *i.e.*, $\mathbf{sat}(C_{2n}, \psi)$. A trust negotiation is *successful* with respect to a client-set trust requirement, ψ, if the server's final disclosure satisfies ψ, *i.e.*, $\mathbf{sat}(C_{2n+1}, \psi)$. When it is implicit whether ψ is client- or server-set, we just say that $\{C_i\}_{i\in[0,2n+1]}$ satisfies ψ. In negotiation strategies where they provide a goal that focuses the credential exchange, we call trust requirements *trust targets*.

In some trust negotiation strategies introduced in Section 5, disclosures are guided by *credential requests* that are also exchanged by the negotiation participants. Credential requests are formalized by a sequence of credential expressions that accompanies the trust negotiation and that has the same length as the sequence of disclosures. For a given trust negotiation, $\{C_i\}_{i\in[0,2n+1]}$, an accompanying sequence of credential requests has the form $\{\psi_i\}_{i\in[0,2n+1]}$.

5. Negotiation Strategy

In our trust negotiation architecture, the negotiation strategy determines the search for a successful negotiation. The strategy controls which credentials are disclosed, when they are disclosed, and which credentials are requested from the other subject to unlock local credentials. Successful trust negotiation is not always possible. One subject or the other may not possess needed credentials, or subjects may govern their credentials by policies that, together, impose cyclic dependencies. The strategy determines when the negotiation instigator—the client in our architecture—gives up on a negotiation.

Some desirable properties of negotiation strategies are as follows. A strategy should lead to a successful negotiation whenever one exists; that is, it should be *complete*. It should terminate with failure when success is impossible. Ideally, it should avoid disclosing any credentials that are not needed for the negotiation to succeed. Finally, a strategy should be reasonably efficient. We analyze the extent to which these properties are satisfied by using the abstract model defined in Section 4.2.

Within the context of the abstract model, we identify each negotiation strategy with a set of trust negotiations. This high level of abstraction focuses our attention on the essential relationships between CAPs and the disclosures and requests that flow between client and server SAs in each strategy. Some practical matters are not formalized here: what is an effective procedure for constructing counter requests; how to truncate negotiations early when success is impossible; and when do service requests flow? We discuss these matters informally.

This section defines and analyzes two negotiation strategies. The first, our *eager* strategy, is complete and efficient. However, it discloses many credentials needlessly. The second, our parsimonious strategy, begins by exchanging credential requests, but no credentials, exploring (backwards) all possible sequences of disclosures that lead to the satisfaction of a given trust target. The sequence of requests reaches a request that can be satisfied by unprotected credentials at exactly the point where the backwards exploration of fruitful disclosure sequences reaches the beginning of the shortest sequences. Starting at that point, the strategy performs a credential exchange that is minimal in length and that makes a locally minimal disclosure at each step. Unfortunately, because of limits on the degree of cooperation between the two participants in this strategy, a globally minimal disclosure is not guaranteed.

5.1. An Eager Strategy

In this simple strategy, two security agents take turns sending each other every credential they have that is currently unlocked. As credentials are exchanged, more credentials become unlocked. The client terminates a negotiation when it receives a set of credentials from the server that it has already received (no new credentials) or the set it receives unlocks no new client credentials.

Definition. A trust negotiation, $\{C_i\}_{i\in[0,2n+1]}$, is in the *eager strategy* if each C_i is the **maximal** set such that **unlocked**$(C_0, \emptyset)$ and **unlocked**(C_{i+1}, C_i) for all $0 \leq i \leq 2n$ and $C_{i+2} \supseteq C_i$ for all $0 \leq i \leq 2n-2$. We call any negotiation in the eager strategy an *eager negotiation*.

The strategy is reasonably efficient: the number of credential exchanges is bounded by the number of credentials each participant has. A tighter bound is the length of the longest chain of dependencies among credentials possessed by the two participants.

Claim 1 (***Efficiency of eager negotiation*****):** The length, $2n+2$, of any eager negotiation, $\{C_i\}_{i\in[0,2n+1]}$, is at most $2\times$**min**(|ClientCreds|, |ServerCreds|).

Claim 2 (***Completeness of eager negotiation*****):** For any credential expression, ψ, if there exists any trust negotiation satisfying ψ, then there exists an eager negotiation satisfying ψ.

For a given target credential expression, such as the SGP of the service, it is not always necessary to perform an entire eager negotiation. To facilitate this optimization, one may choose to have the server return the SGP as ψ_1. One may modify the eager strategy slightly, making the first two disclosures empty, allowing the service request and SGP to flow first. After those first two messages, if the client has unprotected credentials that satisfy the SGP, it discloses them with a repeat of the service request. If the client does not have such credentials, it terminates the negotiation without sending any credentials. If the client has satisfactory but locked credentials, it continues the negotiation by sending its unlocked credentials and requesting the server's unlocked credentials, checking at each round whether the SGP can now be satisfied with its unlocked credentials. Eventually, either the SGP can be satisfied—in which case the client resends the service request along with the required credentials—or the client determines that negotiation has failed.

An alternative deployment does not use mobile policies: the SGP is never sent to the client. With each of its disclosures, the client repeats the service request, until either service is granted, or no new credentials are disclosed and the negotiation fails.

The strengths of the eager strategy are its simplicity and the fact that no information about policies or credentials is disclosed except where CAPs are satisfied. Its weakness is that it discloses credentials without regard to their relevance to the present negotiation: there is no provision for disclosing on a need-to-know basis.

5.2. A Parsimonious Strategy

Eager negotiations begin exchanging credentials essentially immediately. They make little or no use of credential requests, since all unlocked credentials are disclosed without regard for those requests. This section defines and analyzes the *parsimonious strategy*, which differs from the eager strategy in these respects. An intuitive explanation precedes the formal definition. The numbering in that intuitive explanation corresponds to the numbering in the formal definition below.

1. Requests are exchanged to guide the negotiation toward satisfying a particular trust target, ψ. A trust target is a specific credential expression, such as a SGP or a trust requirement set by the client. To simplify the presentation, we assume the trust target is a SGP. The specification and ensuing results can easily be generalized to cover the case of a trust target set by the client as well. Under this assumption, the first credential request from the server is the trust target.
2. When and if a request is sent that can be satisfied by unprotected credentials, those credentials are disclosed in the next stage.

3. Initial credential disclosures are all empty, until a credential request occurs that can be satisfied by unprotected credentials. If no request is ever sent that can be satisfied by unprotected credentials, the negotiation terminates without disclosing any credentials.
4. Each successive credential request is derived from its predecessor in a manner that makes satisfying that request a necessary and sufficient condition for a disclosure to unlock credentials that satisfy the predecessor.
5. After a request is satisfied by a disclosure of unprotected credentials, the client then resends its prior requests, going through them backwards, at the same time disclosing appropriate credentials to unlock solutions to those requests.
6. As mentioned above in point 2, when and if a request is sent that can be satisfied by a set of unprotected credentials, a **minimal** such set is disclosed in the next stage. Each successive step also discloses a minimal credential set that satisfies a credential request, working backwards through the requests that occurred thus far. The client, which drives the negotiation, will have recorded each of the requests that has flowed. It refers to requests it received from the server when selecting credential disclosures; it resends the requests it sent to the server, as outlined in point 5 above. Each disclosure unlocks the next, until a disclosure satisfying the original trust target is unlocked.

Definition. Let the target credential expression be y. To be a *parsimonious negotiation with respect to the trust target*, y, a trust negotiation, $\{C_i\}_{i\in[0,2n+1]}$, must be accompanied by a sequence of credential requests, $\{y_i\}_{i\in[0,2n+1]}$ and must satisfy the following:

1. $y = y_1$.
2. If there exists a j, 0 [j [$2n$+1, and a C, C ` ClientCreds or $C$ ` ServerCreds, such that **sat**(C, y_j) and **unlocked**(C, ∅), then let k be the least such j and fix $\underline{C}$ to be a **minimal** corresponding C. Otherwise, let $k = 2n+1$.
3. For all i, 0 [i [k, $C_i = \emptyset$.
4. For all i, 0 [$i < k$, y_i and y_{i+1} have the following relationship:
   For all C, with respectively C ` ClientCreds or $C$ ` ServerCreds, we have **sat**(C, y_{i+1}) *iff* there exists a C¢ with respectively C¢` ServerCreds or $C$¢` ClientCreds (opposite of C), such that **sat**(C¢ y_i) and **unlocked**(C¢C).
5. Prior client requests (which have even indexes) are replayed. If k is even, we require $y_{k+j} = y_{k-j}$ for even j, 2 [$j < 2n-k$ (if any). If k is odd, we require $y_{k+j} = y_{k-j}$ for odd j, 1 [$j < 2n-k$ (if any).
6. If $k < 2n+1$, we require that $C_{k+1} = \underline{C}$, and for all j, 1 [$j < 2n-k$, (if any) C_{k+j+1} is a **minimal** set satisfying **sat**(C_{k+j+1}, y_{k-j}).

In the following discussion, ClientCreds, ServerCreds, gov$_{\text{client}}$, and gov$_{\text{server}}$ are fixed but arbitrary.

Claim 3 (*Determinacy of Requests in Parsimonious Negotiation*): Given two different parsimonious negotiations with the same trust target and with associated credential requests given by $\{y_i\}_{i\in[0,2n+1]}$ and $\{y'_i\}_{i\in[0,2m+1]}$, we have y_i h y'_i for all i, 1 [i [k, where k is either the least index such that y_k can be satisfied by unprotected credentials, $2n$+1, or $2m$+1, whichever is least.

Claim 4 (*Correctness of Parsimonious Negotiation*): Let $\{C_i\}_{i\in[0,2n+1]}$ be a parsimonious negotiation with respect to trust target y with associated credential requests given by $\{y_i\}_{i\in[0,2n+1]}$. If there exist k and C with 1 [k [$2n+1$ and (C ` ClientCreds or $C$ ` ServerCreds) such that **sat**(C, y_k) and **unlocked**(C, ∅), then there exists a parsimonious negotiation $\{C$¢$\}_{i\in[0,2m+1]}$ that satisfies y and that is the same as $\{C_i\}_{i\in[0,2n+1]}$, except for possibly being longer.

Claim 5 (*Completeness and Efficiency of Parsimonious Negotiation*): If there exists a trust negotiation that satisfies y, then (1) every sufficiently long parsimonious negotiation with trust target y also satisfies y and (2) no parsimonious negotiation with trust target y has length greater than 4×**min**(|ClientCreds|, |ServerCreds|).

Any deployment of the parsimonious strategy should take advantage of the fact that, if a successful negotiation exists, the initial exchange of credential requests will encounter a request that can be satisfied by unprotected credentials within the first 2m requests, where m = |ClientCreds|. If such a request has not occurred, the negotiation should be terminated.

An issue faced deploying this strategy is effective derivation of credential requests that satisfy the requirements stated in the construction. As long as the credential language is able to represent logical "and" and "or," these counter requests can be constructed. A "brute force" approach constructs counter requests as follows. (A more efficient approach for the credential expression language presented in Section 6 can be constructed. It is not presented here due to space constraints.) Assume that . and - denote "and" and "or," *i.e.*, that for all C, we have **sat**(C, .$\{y_0, y_1, \ldots, y_m\}$) *iff* **sat**($C$, y_i) for each i, 1 [i [m and **sat**(C, -$\{y_0, y_1, \ldots, y_m\}$) *iff* **sat**($C$, y_i) for some i, 1 [i [m. Counter requests can be computed by the server from the incoming request by using $\mathrm{counter}_{\mathrm{server}}(y)$ = **-**{$\mathrm{gov}_{\mathrm{server}}(C)$ | **sat**(C, y)}, where $\mathrm{gov}_{\mathrm{server}}(C)$ = **.**{$\mathrm{gov}_{\mathrm{server}}(c)$ | $c \in C$}. Counter requests can be computed analogously by the client.

Example. We illustrate the parsimonious strategy by ignoring several issues. We ignore the fact that credentials are submitted with supporting credentials. We ignore the internal structure of credentials. We use an unrealistic credential expression language that treats credentials as propositional variables.

Let ClientCreds={*a*, *b*, *c*, *d*}, ServerCreds={*x*, *y*}, $\mathrm{gov}_{\mathrm{client}}$={*a* **hÛ**, *b* **hÛ**, *c* **h***x*, *d* **h***y*}, and $\mathrm{gov}_{\mathrm{server}}$={*x* **h***a*-*b*, *y* **h***a*-*b*}. In the exchanges, we use z to represent values that are not determined or used by the parsimonious negotiation. Recall that y_1 represents the trust target—an SGP: y_o=z, y_1=(*a* .*d*)-(*c* .*b*), y_2=*x*-*y*, y_3=*a*-*b*, y_4=*x*-*y*, y_5=z, y_6=z, y_7=z; C_o=**Û**, C_1=**Û**, C_2=**Û**, C_3=**Û**, C_4={*a*}, C_5={*x*}, C_6={*b*,*c*}, C_7=z. The total set of credentials disclosed by the client in this negotiation is not minimal, because the client could have disclosed *b* in C_4 instead of *a*. However, if the client had tried this, the server choice in C_5 could instead have been to disclose {*y*}, forcing the client to disclose {*a*,*d*} in C_6, which again is not minimal. This example illustrates the difficulty of devising a negotiation strategy in which the two participants cooperate in exchanging a globally minimal set of credentials.

5.3. Hybrid Strategies

This section sketches informally an approach to combining eager and parsimonious strategies in an effort to use each with the credentials for which it is better suited. CAPs can be made two-part, comprising not only a credential expression, but also a flag to select between parsimonious and eager disclosure. A credential flagged for eager disclosure would be disclosed freely to all sites that present credentials that satisfy the credential-expression component of its CAP. One flagged "parsimonious" would be disclosed only as part of a locally minimal exchange and successful negotiation. A hybrid strategy begins with a phase that uses an eager strategy to attempt to negotiate using only credentials flagged for eager disclosure. If success is possible using just those credentials, the negotiation succeeds during the eager phase. If not, phase two uses a parsimonious strategy to attempt to establish trust by using all credentials. Phase two takes advantage of credentials exchanged during phase one. In a hybrid negotiation, the client determines when phase one has completed unsuccessfully and phase two begins. The client indicates in each request to the server which strategy it is currently employing.

6. Credential Expression Languages

Credential expressions could specify combinations of credentials directly, for instance by representing credential requirements as Boolean combinations of specific credentials. However, by layering levels of abstraction, we aim to provide a basis for overcoming the complexity of authenticating strangers by using credentials obtained for other purposes. Deriving relevant properties from such credentials requires care and expertise. It is important to separate this task from that of defining access requirements for individual services and credentials. For this reason, our credential expression language comprises two parts, separating authentication from authorization. For authentication, credentials are mapped to roles; for authorization, access requirements are expressed as combinations of roles.

The language we present is incomplete and not a formal language-design proposal. However, two language features, introduced and discussed in this section, make or reiterate important points that will guide further work in this area. First, role attributes enhance expressiveness. Second, the monotonic relationship between credentials and access is essential in any context where a subject can withhold disclosure of its own credentials.

This section presents our *Role-based Authorization Language* (*RAL*) used in the example negotiation presented in Section 7. An *authorization policy* defined in RAL consists of a *role-constraint expression*, which expresses requirements for access to the service or credential that it governs. These requirements are expressed in terms of roles of the subject seeking access. These roles are defined by an *authentication policy* written in our *Property-based Authentication Language* (*PAL*), which is also presented in this section. A PAL authentication policy assigns a subject to roles based on subject properties derived from credentials owned by the subject and from the roles of the issuers of those credentials. This assignment is independent of the question of the subject's access to the service. Thus, a PAL role is not a capability, but represents a derived property of the subject.

Section 6.1 introduces PAL, which maps credentials to roles. Section 6.2 then introduces RAL, which uses those roles to define access requirements. Section 6.3 discusses the significance and impact of supporting role attributes.

6.1. Property-based Authentication Language

A PAL authentication policy defines one or more roles. A role is a property of subjects defined in terms of the credentials they possess and the attribute values of those credentials. The notion of assigning subjects to

```
hasCredit(Amount) ¬  LetterOfCredit : credit,
     creditor(LetterOfCredit.issuer), Amount = LetterOfCredit.amount.
creditor ¬  Creditor : creditor, self(Creditor.issuer).
knownClient() ¬  Ref : reference, self(Ref.issuer),
     Ref.relationship = “shipping client”.
knownClient() ¬  Ref1 : reference, Ref2 : reference,
     Ref1.issuer  ¹  Ref2.issuer, shipper(Ref1.issuer), shipper(Ref2.issuer),
     Ref1.relationship = “shipping client”,  Ref2.relationship = “shipping client”.
shipper() ¬  Ref : reference, self(Ref.issuer), Ref.relationship = “shipper”.
shipper() ¬  Ref1 : reference, Ref2 : reference, Ref1.issuer ¹  Ref2.issuer,
     knownClient(Ref1.issuer), knownClient(Ref2.issuer),
     Ref1.relationship = “shipper”, Ref2.relationship = “shipper”.
```

Figure 3. Six PAL Rules

roles based on credentials goes back to Bina et al. [1]. In Seamons et al. [13] policies are Prolog programs. However, with the exception of a few features, PAL is based on the Trust Policy Language (TPL) of Mass et al. [10]. TPL introduced two intertwined innovations, which we adopt: the role is TPL’s sole procedural abstraction; TPL assigns a role not only to the submitting subject, but also to each owner of one of the supporting credentials.

PAL’s role attributes enhance TPL, as discussed in Section 6.3. PAL also differs from TPL in that roles are *monotonic* functions of credentials: disclosing more credentials cannot decrease the roles to which a subject authenticates. This is a requirement in trust negotiation because subjects have control over the credentials they disclose, but not the credentials they obtain.

For presentation purposes, we give PAL a concise notation derived from syntax of constraint logic programming languages. A PAL role is a user-defined predicate over the public keys of subjects. Role attributes take the form of additional parameters. A PAL authorization policy consists of a set of *rules*, each of which has the following form:

role(Attributes, ...) ¬ CredentialVar :
credentialType, ...,
role$_1$(credentialVar.issuer, Attributes$_1$), ...,
credentialConstraints, ...

The *head* of the rule, to the left of the arrow, consists of a single role, applied to an implicit subject key and to zero or more explicit role attributes, illustrated here by **Attributes**. Together, the rules with the same role in their head define that role. The *body* of the rule comprises the portion of the rule to the right of the arrow. One or more credential variables are introduced, each by a variable type declaration, as illustrated by **CredentialVar : credentialType** The attributes of a credential are denoted with the syntax **CredentialVar.attribute.** Credential variables and role attributes have names that start with capitol letters. Roles, credential types, and credential attributes have names that start with lowercase letters.

Within a given rule, the subject being categorized owns each credential denoted by a credential variable in that rule. The rule states that the subject can authenticate to the role **role** with role attributes **Attributes** if the subject owns a combination of credentials of the designated types such that each of the rule’s requirements holds. These requirements are of two kinds. The first requires that the issuer of a credential belongs to a given role. Such an *issuer-role constraint* is an application of a role to a credential-issuer key and role attribute variables, and is illustrated by **role$_1$(CredentialVar.issuer, attributes$_1$)**. Note that the subject key is identified explicitly in role applications that appear in a rule body. Each credential variable must appear in exactly one issuer-role constraint. The second kind of requirement is a *general constraint* on the attributes of the credentials and roles appearing in the rule. These are typically relational expressions, using operators such as =, g, and [. Although not specified in this paper, user defined functions and operators may also be used. In addition to imposing requirements on the credentials denoted by the credential variables, general constraints serve to define the attributes of the role in the rule head.

Figure 3 shows six PAL rules. The first rule defines one way a subject can be shown to be in the **hasCredit** role. If the full authentication policy contained other rules that defined **hasCredit**, those rules would state other ways to authenticate to the role. The rule here states that a subject is in the **hasCredit** role if it owns a **credit** credential issued by a subject in the **creditor** role. The rule also states that role attribute, **Amount**, is defined by credential attribute, **amount.**

To be satisfied, the rule must be evaluated in the presence of the **credit** credential mentioned above, as well as credentials, owned by the **credit** credential’s issuer, that satisfy a rule defining the **creditor** role. The

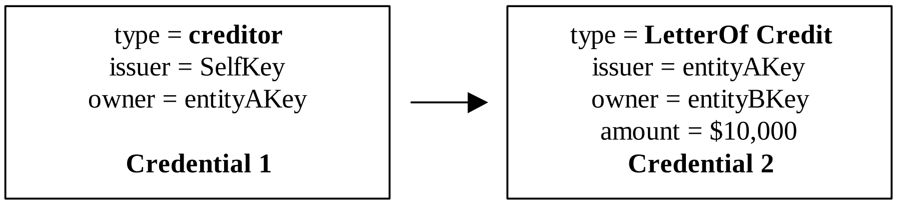


Figure 4. A chain that proves subject A is in the **creditor** role and subject B is in the **hasCredit** role, as those roles are defined in Figure 3

```
reputation(Rating) ¬  Membership : businessOrgMember,
     businessOrganization(Membership.issuer), Rating = Membership.rating.
businessOrganization()¬  ....
clientWithAccount(AccountNumber)¬  Account : account,
     self(Account.issuer), AccountNumber = Account.number.
newShippingClient(Pickup, Delivery)¬  Destiniation : contract, Warehouse : lease,
     knownBusiness(Destination.issuer), warehouseOwner(Warehouse.issuer),
     Pickup = Warehouse.location, Delivery = Destination.deliveryLocation.
knownBusiness()¬  ....
warehouseOwner()¬  ....
```

Figure 5. Role definitions that, together with those in Figure 3, compose the example's authentication policies. We present the authentication policies defined by the client and the server together here, as some rules are used by both subjects. Note that three rules are incomplete. Their content and the credentials needed to satisfy them are elided from the example presentation to save space.

second rule shown in Figure 3 is such a rule. It states that a subject is in the **creditor** role if it owns a **creditor** credential issued by a member of the **self** role. The **self** role is unlike other roles. It is predefined and its sole member is the owner of the policy, referred to as *self*. A credential issued by the policy owner is a *self-signed credential*, as introduced by Mass et al. Unlike other credentials, self-signed credentials are part of an authentication policy; they define the valid roots of all credential chains. A chain of credentials that supports membership in the **hasCredit** role is shown in Figure 1

By making use of the TPL notion of role as a procedural abstraction mechanism, an authorization policy can trace chains of unbounded length [10]. Rules 3 through 6 in Figure 3 define two roles, **knownClient** and **shipper**. A subject is in the **knownClient** or **shipper** role if it has a corresponding self-signed **reference** credential from the policy owner. A subject is also in the **knownClient** role if it has a reference from two **shipper** members. Similarly, a subject is also in the **shipper** role if it has a reference from two **knownClient** members.

To summarize, an authentication policy has three parts: a set of rules, a set of self-signed credentials, and a set of user defined functions. In this paper we generally exhibit only the rules. However, in any mobile policy, all three parts must flow.

6.2. Role-based Authorization Language

This section introduces the Role-based Authorization Language (RAL). A RAL authorization policy comprises a role-constraint expression, together with a PAL authentication policy defining each role used therein. A role-constraint expression is a logical formula consisting of role applications and attribute constraints combined with the logical connectives AND and OR. (These connectives give exactly the monotonic formulas discussed above.) A role application consists of a role name applied to a subject (in the example, just one of the reserved words **Client** or S**erver**) and role attribute variables. Attribute constraints are arbitrary constraints on the role attributes appearing in the rule. An authorization policy that governs a service may also use service parameter names in these constraints. As in PAL, user defined functions may be used.

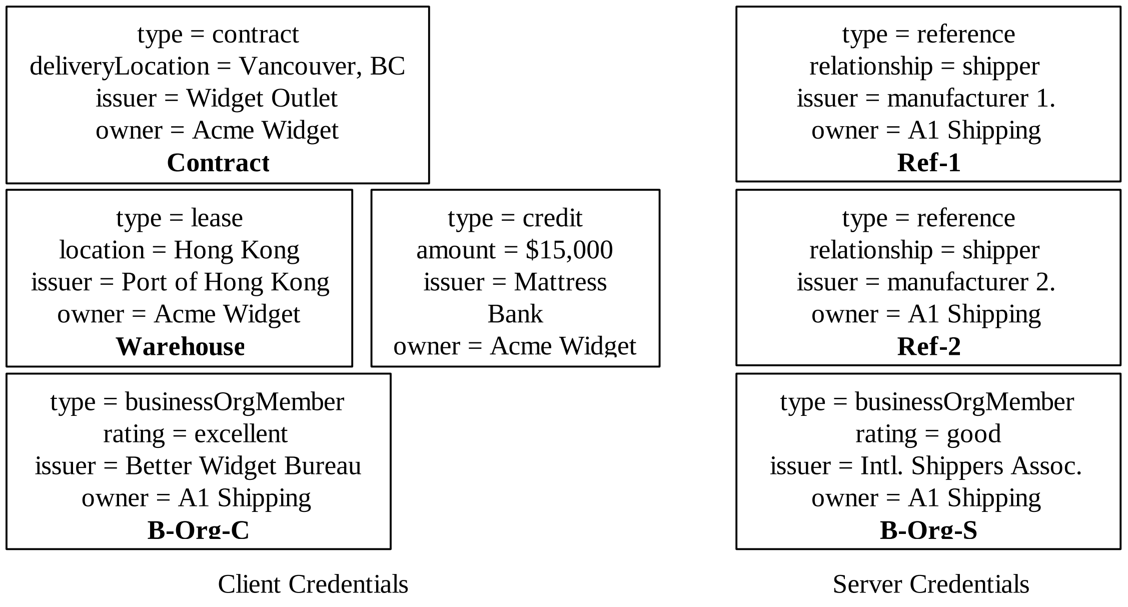
Example. The following authorization policy could govern a request for a service that schedules shipping. The parameters of the request are **PickupLocation**, **Destination**, and **Tons**.

hasCredit(Client, Amount) AND
 Amount ³ Tons ´
 costPerTon(PickupLocation, Destination)

This policy requires that the client be in the **hasCredit** role with the **Amount** attribute value at least sufficient to

```
clientWithAccount(Client, AccountNumber) OR
(reputation(Client,Rating) AND Rating Î {good, excellent} AND
    (knownClient(Client) OR
        (newShippingClient(Client, Pickup, Delivery) AND
            PickupLocation = Pickup AND Destination = Delivery)) AND
    hasCredit(Client, Amount) AND Amount ³ Tons ´ costPerTon(PickupLocation, Destination))
```

Figure 6. Policy governing the service, “Schedule Shipping.” Parameters of the service request include: PickupLocation, Destination, and Tons.



type = contract
deliveryLocation = Vancouver, BC
issuer = Widget Outlet
owner = Acme Widget
Contract

type = lease
location = Hong Kong
issuer = Port of Hong Kong
owner = Acme Widget
Warehouse

type = credit
amount = $15,000
issuer = Mattress Bank
owner = Acme Widget

type = businessOrgMember
rating = excellent
issuer = Better Widget Bureau
owner = A1 Shipping
B-Org-C

Client Credentials

type = reference
relationship = shipper
issuer = manufacturer 1.
owner = A1 Shipping
Ref-1

type = reference
relationship = shipper
issuer = manufacturer 2.
owner = A1 Shipping
Ref-2

type = businessOrgMember
rating = good
issuer = Intl. Shippers Assoc.
owner = A1 Shipping
B-Org-S

Server Credentials

Figure 7. The credentials owned by each negotiation participant. Each credential is labeled by a name used to refer to it in the example.

pay the cost of the desired shipping. The function **costPerTon** is user defined.

When an authorization policy is combined with an authentication policy that defines all the roles it uses, the two together define a set of *solutions*. Each solution is a minimal set of credentials that proves that the subject in question (client or server) belongs to the required roles with the required attributes.

6.3. Role Attributes

The association between attributes and roles enables the policy writer to make credential attributes, exported as role attributes, available in authorization policies. This enables authorization to depend on attributes—such as credit line, rating, or age—without the inconvenience of defining special purpose roles that impose the constraints within the authentication policy.

Role attributes also have at least two other advantages in the context of TPL-style role definitions. First, by exporting credential attributes, they enable attributes of different credentials in a chain to be compared. For instance, they could be used to compare attributes of a credential owner with attributes of the credential's issuer. Second, role attributes can be used to keep track of and to bound the length of a credential chain. Consider the **shipper** role defined in Figure 3. While it is very convenient to let shippers and shipping clients vouch for each other, as the number of subjects in a chain goes up, the policy writer's confidence in the last subject's role may go down. An attribute, **ChainLength**, can be added to each of **shipper** and **knownClient**. By initializing it to zero in the rules that use self-signed credentials, and incrementing it in each of the (mutually) recursive rules, the length can be computed. An authorization policy can then easily impose an upper bound on the length of acceptable chains.

7. Example Negotiations

This section presents two example trust negotiations based on the eager and parsimonious strategies. In the

Client Credential Access Policies:

Contract$_C$ h shipper(Server) AND reputation(Server, Rating) AND RatingÎ {good, excellent}
Credit$_C$ h reputation(Server, Rating) AND RatingÎ {good, excellent}
Warehouse$_C$ h shipper(Server)
B-Org-C$_C$ h true

Server Credential Access Policies:

Ref-1$_S$ h reputation(Server, Rating) AND RatingÎ {good, excellent}
Ref-2$_S$ h reputation(Server, Rating) AND RatingÎ {good, excellent}
B-Org-S$_S$ h true

Figure 8. The CAPs that govern each credential in the example. Only the authorization portion of each policy is shown. The authentication portion comprises the relevant rules from Figure 3 and Figure 5. The policy by which the client and server, respectively, govern a credential is designated by the credential's name with subscript C and S, respectively.

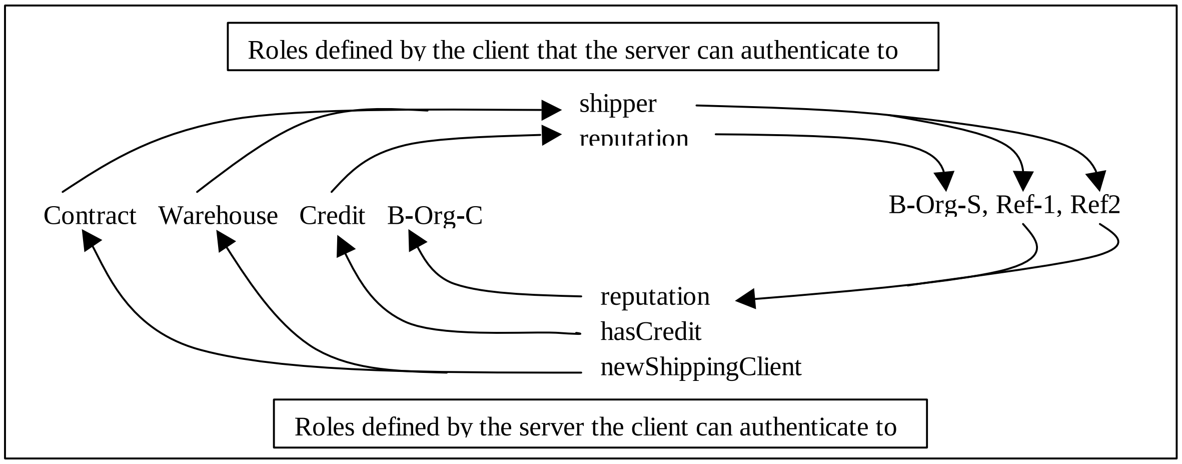

Roles defined by the client that the server can authenticate to
shipper
reputation
Contract
Warehouse
Credit
B-Org-C
B-Org-S, Ref-1, Ref2
reputation
hasCredit
newShippingClient
Roles defined by the server the client can authenticate to

Figure 9 A graph showing the client's credentials to the left and the server's credentials to the right. Incoming arcs to the credentials show the dependency to roles the credential owner can authenticate to using the credentials. Outgoing arcs show the dependency to roles that govern access to the credential. Credentials with no outgoing arc are freely available. Only the roles actually used in the example negotiation are shown.

examples, a negotiation is undertaken to authorize on-line scheduling of shipping services by a small, fictitious widget manufacturer, Acme Widget. The negotiation successfully establishes the trust required to authorize the service request and schedule shipping.

Figure 3 and Figure 5 present the role definitions used by the authentication portions of the SGP and CAPs, as well as the requests for credentials that may flow during the negotiation. To save space, we list each rule only once, although it may have to appear in several places in the various policies and requests. Although not shown, the authentication policies include necessary self-signed credentials. In particular, the server's authentication policies that define **creditor** also include a self-signed **creditor** credential owned by Mattress Bank**.** Similarly, the client's authentication policies that define **knownClient** also include two self-signed **reference** credentials, one owned by manufacturer 1 and one by manufacturer 2, both with **relationship = "shipping client"**.

Figure 6 presents the authorization policy that governs the service requested. Figure 7 shows the client and server credentials. Figure 8 shows the CAPs of each of those credentials. Figure 9 shows the dependencies between credentials, roles governing access to those credentials, and credentials that authenticate their owner to a given role.

For the negotiation based on the eager strategy, the six stages of the negotiation are described below. Following the initial request for service in stage 1, each stage shows the unlocked credentials that are disclosed.

Stage 1. Client sends (via client SA) request to schedule shipping 5 tons of cargo.

Stage 2. Server SA sends **B-Org-S** to client.

Stage 3. Client SA repeats request to schedule shipping and sends **B-Org-C**, **Credit** to server.

Stage 4. Server SA sends **B-Org-S**, **Ref**$_1$, and **Ref**$_2$.

Stage 5. Client SA repeats service request and sends **B-Org-C**, **Credit, Contract,** and **Warehouse**.

Stage 6. Server SA receives service request and authorizes it, as attached credentials satisfy the SGP.

For the negotiation based on the parsimonious strategy, the eight stages of the negotiation are described below. For each stage where an incoming request for credentials is received, the description shows the solution to the request (a list of local-site credentials) that is selected by the SA. Credentials that are locked appear underlined in the solution. Counter requests combine the CAPs of these locked credentials

Stage 1. Client sends (via client SA) request to schedule shipping 5 tons of cargo.

Stage 2. Server SA returns the SGP.

Stage 3. Client SA's solution to incoming request: **Contract**, **Warehouse**, **Credit**. Outgoing request for credentials: **reputation(Server, Rating) AND Rating c {good, excellent} AND shipper(Server).**

Stage 4. Server SA's solution to incoming request: **B-Org-S**, **Ref**$_1$, **Ref**$_2$. Outgoing request for credentials: **reputation(Client, Rating) AND Rating c {good, excellent}.**

Stage 5. Client SA's solution to incoming request: **B-Org-C.** Outgoing request for credentials (repeating request sent in Stage 3): **reputation(Server, Rating) AND Rating c{good, excellent} AND shipper(Server).** Outgoing credentials (which solve the request sent by server SA in Stage 4): **B-Org-C.**

Stage 6. Server SA's solution to incoming request: **B-Org-S**, **Ref**$_1$, **Ref**$_2$. Outgoing request for credentials: **None.** Outgoing credentials: **B-Org-S**, **Ref**$_1$, **Ref**$_2$

Stage 7. Client SA repeats service request from Stage 1, attaching credentials that solve the SGP sent by the server SA in Stage 2. Outgoing credentials: **Contract**, **Warehouse**, **Credit.**

Stage 8. Server SA receives service request and authorizes it, as attached credentials satisfy the SGP.

8. Related Work

Credential-based authentication and authorization systems can be divided into three groups: identity-based, property-based, and capability-based systems. The original, public key certificates, such as X.509 [19] and PGP [17], simply bind keys to names (although X.509 version 3 certificates later extended this binding to general properties). Such certificates form the foundation of identity-based systems, which authenticate a subject'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. Property-based credentials were introduced by Bina et al. to incorporate the binding of arbitrary attributes. Trust establishment between strangers can be based on the properties of the subjects without requiring familiarity with the actual subjects. Systems have emerged recently that use property-based credentials to manage trust in decentralized, distributed systems [6][10][13][16]. Capability-based systems [2][3][4][18], discussed further below, manage delegation of authority to operate on a particular application. Capability-based systems are not designed for establishing trust between strangers, since clients are assumed to possess credentials that authorize specific actions with the application server.

Winslett et al. [16] focus on establishing trust between strangers. They present an architecture for using credentials to authorize access to distributed resources. Client and server security assistants manage both the credentials and the policies governing access to sensitive resources. They emphasize the need for credential and policy exchange with little intervention by the client. Seamons et al. [13] continue in this vein, developing policies written in Prolog that use credentials and credential attributes to authenticate clients to roles that have attributes, which can be used in authorization decisions. This work addresses credential sensitivity by using mobile policies to support private client selection of credentials to submit for authorization.

Johnston et al. [6] use both attribute certificates (property-based credentials) and use-condition certificates (policy assertions) to determine 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 whether all use conditions are met. The certificates are assumed not to be sensitive.

The Trust Establishment Project at the IBM Haifa Research Laboratory [10][11] has developed a system for establishing trust between strangers according to policies that specify constraints on attribute-contents of public-key certificates. Servers use a collector to gather supporting credentials from issuer sites. The system assumes that credentials are not sensitive. The companion Trust Policy Language (TPL) is a special-purpose logic programming language, with XML syntax, that maps certificate holders to roles. TPL policies also map the issuers of each supporting credential to a role. These roles can be used by existing role-based access control mechanisms. The example negotiations presented in Section 7 use a simplified language based on TPL.

The capability-based KeyNote system of Blaze et al. [2][3][4] manages delegation of authority. A KeyNote credential is an assertion that describes the conditions under which one principal authorizes actions requested by other principals. A *policy* is also an assertion that delegates authority on behalf of the associated application to otherwise untrusted principals. Thus an application's policy defines the root of all delegation chains. KeyNote credentials express delegation of authority 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 derive roles from credential attributes, or otherwise bridge the divide between the application and credentials that were issued for unrelated purposes. The IETF Simple Public Key Infrastructure [18] uses a similar approach to that of KeyNote by embedding authorizations directly in certificates.

SSL [6], the predominant credential-exchange mechanism in use on the web today, and its successor TLS [6][7], support credential exchange during client and server authentication. There is no opportunity for the server to authenticate any information about the client before disclosing the server's credential. That is, sensitive server credentials cannot be protected. Furthermore, if the credential disclosed by the server does not satisfy the client, the client has no opportunity to request additional credentials from the server. This presents a serious problem when the client and server are strangers: it is unlikely that any single issuer would be an acceptable authority on all server attributes of interest to all potential clients.

9. Conclusions and Further Work

We have presented a model and an architecture for incrementally establishing mutual trust between clients and servers. The architecture solves the problem of establishing trust sufficient to allow disclosure of sensitive credentials that must be exchanged to authorize application-level operations.

We have discussed two negotiation strategies, one eager and one parsimonious. The eager strategy negotiates efficiently, succeeding whenever possible. It does not exchange credential requests, or otherwise attempt to minimize credential disclosures. There is an advantage in this: credential requests can disclose sensitive information. Credential requests exchanged in the parsimonious strategy can reveal a great deal about which credentials—and properties—a subject has. Although the eager strategy may reveal a credential unnecessarily, it does so only in accordance with the CAP.

The parsimonious strategy conducts a minimal-length exchange in which each disclosure is a locally minimal set. It does this by conducting an exchange of credential requests that in effect considers every possible successful exchange. It remains open how to ensure that the union of each participant's disclosures is minimal. Achieving global minimality remains a goal.

The strategies presented here assume that both participants cooperate in using the strategy. Further research is required to determine whether and how that assumption can be relaxed. A parsimonious negotiation guarantees locally minimal credential disclosure only when both parties "bargain in good faith." This means that each SA assumes the following about the other SA. When the other SA responds to an incoming request by issuing a counter request, if that SA subsequently receives credentials that satisfy the counter request, together with a repetition of the original request, it will return credentials that satisfy that original request. One advantage of a hybrid negotiation strategy, such as the one sketched in Section 5.3, is that the eager phase could be used to establish trust that the other negotiation partner will bargain in good faith before entering a parsimonious negotiation.

Further research is also needed in dimensions of credential-governing policy that determine or influence negotiation strategy. In addition to the credential expression whose satisfaction unlocks credential disclosure, additional policy content may assist in determining credential disclosures and requests. Two orthogonal issues that a credential-governing policy might address are as follows. The first is whether voluntary credential disclosure is permitted whenever the credential is unlocked or whether the credential must first be explicitly requested (presumably implying that its disclosure is necessary for successful negotiation). The second is whether it is permitted to issue counter requests when the credential is requested, potentially disclosing that the credential is held without disclosing the credential itself. The hybrid strategy presented in Section 5.3 assumes that each credential either can be voluntarily disclosed or can have a counter request issued in an effort to unlock it. A natural generalization captures the other two possibilities: that a credential (1) can *both* be voluntarily disclosed and have counter requests issued and (2) can *neither* be voluntarily disclosed nor have counter requests issued. It seems intuitive that some credentials would naturally fall into each of these categories, based, for instance, on whether possession or contents of the credential were more sensitive, and the reasons for the sensitivity.

The potential impact of these policy concepts on hybrid negotiation strategies needs to be examined further. Credentials in the "both" category might enable flexible and focused strategies. Strategies are needed that manage disclosure of credentials whose highly sensitive nature requires the "neither" category. These credentials should only be disclosed to a partner that requests them and that simultaneously provides sufficient credentials to unlock them. For instance, this could be accomplished by a hybrid strategy with an eager negotiation phase followed by a single request for the most sensitive credentials.

Another policy generalization would group credentials into ordered sensitivity classes. This captures a notion of relative sensitivity not captured by other policy constructs above. Strategies could prefer disclosing credentials in lower sensitivity classes.

The architecture presented here addresses some, but not all, of the client's trust needs. It addresses the client's need for trust that enables it to disclose credentials to the server. It does not address the client's need for trust before it requests service. The architecture could assist in negotiating the server's disclosure of credentials that would establish that trust. However, further work is needed to address the question of how the client formulates its request for those credentials. Such a request could be based, for instance, on transaction type or on the contents of service request parameters.

In addition to the above, ongoing work also includes deployment of trust negotiation in a demonstration prototype that uses the eager strategy to negotiate trust between Web clients and servers.

Acknowledgements

The authors wish to thank Marianne Winslett and Sushil Jojodia for their suggestions for improving earlier drafts of this paper. This research project has been supported by DARPA and AFRL under contract F30602-98-C-0222.

10. References

[1] E. Bina, V. Jones, R. McCool, and M. Winslett, "Secure Access to Data Over the Internet," Proceedings of the Third ACM/IEEE International Conference on Parallel and Distributed Information Systems, Austin, Texas, Sept., 1994.

[2] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust Management System," work in progress, Internet Draft, March 1999.

[3] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "KeyNote: Trust Management for Public-Key Infrastructures," Cambridge 1998 Security Protocols International Workshop, England, 1998.

[4] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," *1996 IEEE Conference on Privacy and Security*, Oakland, 1996.

[5] N. Ching, V. Jones, and M. Winslett, "Authorization in the Digital Library: Secure Access to Services across Enterprise Boundaries," *Proceedings of ADL '96 --- Forum on Research and Technology Advances in Digital Libraries*, Washington, DC, May 1996. Available at http://drl.cs.uiuc.edu/security/pubs.html.

[6] T. Dierks, C. Allen, "The TLS Protocol Version 1.0," draft-ietf-tls-protocol-06.txt, Nov. 12, 1998.

[7] S. Farrell, "TLS Extensions for Attribute Certificate Based Authorization," draft-ietf-tls-attr-cert-01.txt, August 20, 1998.

[8] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol," Netscape Communications Corp., Nov., 1996.

[9] W. Johnston, S. Mudumbai, and M. Thompson, "Authorization and Attribute Certificates for Widely Distributed Access Control," *Proceedings of the IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises—WETICE '98*.

[10] Y. Mass, A. Herzberg, D. Naor, Y. Ravid, and J. Mihaeli, "Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers," forthcoming, IBM Haifa Research Laboratory (http://www.hrl.il.ibm.com/), email contact: YOSIMASS@il.ibm.com.

[11] "The Trust Policy Language," IBM Haifa Research Laboratory (http://www.hrl.il.ibm.com/), email contact: YOSIMASS@il.ibm.com.

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

[13] K. Seamons, W. Winsborough, and M. Winslett, "Internet Credential Acceptance Policies," *Proceedings of the 2nd International Workshop on Logic Programming Tools for Internet Applications.* Leuven, Belgium, July, 1997. Available at http://clement.info.umoncton.ca/~lpnet/proceedings97/

[14] W. Winsborough, K. Seamons, and V. Jones, "Negotiating Disclosure of Sensitive Credentials," Second Conference on Security in Communication Networks '99, Amalfi, Italy, Sept., 1999.

[15] W. Winsborough, K. Seamons, and V. Jones, "Automated Trust Negotiation: Managing Disclosure of Sensitive Credentials," Transarc Research White Paper, May 1999.

[16] M. Winslett, N. Ching, V. Jones, and I. Slepchin, "Using Digital Credentials on the World-Wide Web," *Journal of Computer Security*, **5**, 1997, 255-267. Available at http://drl.cs.uiuc.edu/security/pubs.html.

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

[18] Simple Public Key Infrastructure (SPKI), http://www.ietf.org/html.charters/spki-charter.html.

[19] International Telecommunication Union, Recommendation X.509 - Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, August 1997.