Negotiating Disclosure of Sensitive Credentials

William H. Winsborough
Kent E. Seamons
Transarc Corporation
{winsboro,seamons}@transarc.com

Vicki E. Jones
Department of Computer Science
North Carolina State University
vej@eos.ncsu.edu

Abstract

The problem considered is credential exchange between strangers when credentials are sensitive. A framework is presented for negotiating the exchange of sensitive property-based credentials that are exchanged to establish trust between clients and servers. The notion of a trust negotiation strategy is introduced and two strategies with very different properties are discussed. This article is excerpted from [WSJ].

1 Introduction

Distributed software entities face the problem of determining one another's trustworthiness. Current mainstream approaches to establishing trust presume that communicating entities are already familiar with one another. There are essentially two approaches based on this assumption. The first is identity-based: identifying an entity is often a sufficient basis for doing business, but only when the parties are known to one another. The second is capability-based. Here, entities must obtain capabilities that are specific to the resources they wish to act upon, again requiring that familiarity be established out of band. Identity- and capability-based approaches both fail when attempting to establish trust between complete strangers. Other solutions must be developed for use in open systems, such as the web, where the assumption of familiarity is invalid.

Property-based *digital credentials* [Bina94] (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 certificates with attributes [x509a, x509b], can authenticate not just the entity's identity, but arbitrary properties of an entity and of its relationships with other entities. Credentials can be presented by a client to support a service request and to form the basis for its authorization.

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 is about automating trust establishment between strangers through credential exchange when credentials are themselves potentially sensitive and subject to access controls. 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, by requiring a user at the client to decide which credentials to submit to each new service.

This paper presents a framework for client-server applications in which client and server each establishes an access policy for each of its credentials. A credential is disclosed only when its access policy is satisfied by credentials previously obtained from the opposing software agent. When an agent needs additional credentials, it can request them. Credentials flow between the

type = reference
relationship = shippingClient
issuer = entityAKey
owner = entityBKey
Credential 1
	type = reference
	relationship = shipper
	issuer = entityBKey
	owner = entityCKey
	Credential 2

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

client and server through a sequence of alternating credential requests and disclosures, which we call a *trust negotiation*. We call the framework the *Trust Negotiation Framework*.

The framework is parameterized by a *negotiation strategy*. The selected negotiation strategy determines key characteristics of the negotiation, such as which credentials are requested by each agent and when the negotiation is halted. Two negotiation strategies are discussed, each with very different properties. The first strategy discloses only credentials that are essential to the negotiation. Because it is stingy with credential disclosure, we call it a *parsimonious* strategy. The second guarantees that negotiation will succeed whenever possible. Because it achieves this guarantee by turning over relevant credentials as early as possible, we call it an *eager* strategy. The eager strategy terminates efficiently; the parsimonious strategy can be made to terminate efficiently without sacrificing minimal credential disclosure. These strategies illustrate the existence of strategies with important theoretical properties. Issues relevant to future work on practical negotiation strategies are discussed.

Section 2 introduces credentials and explains how they can be used to establish trust between strangers. Section 3 explains why prior systems were not fully automatic in the presence of sensitive credentials. Section 4 presents the Trust Negotiation Framework. Section 5 discusses the parsimonious and eager negotiation strategies. Section 6 discusses related work. Section 7 draws conclusions about design of practical negotiation strategies.

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 [Schneier96]. 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 entity. 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 entity, the issuer of the first credential in the chain (e.g., entity A in Figure 1), to the *submitting entity*, in whom trust is needed. The submitting entity is the owner of the last credential in the chain (e.g., entity C) and can demonstrate ownership of that credential, as outlined above. *Supporting* credentials are owned by entities with whom the submitting entity has a direct or indirect relationship and, although they are not owned by the submitting entity, 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 anyone to verify reliably 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 entity and the known entity 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 entity and its relationships with known entities.

A *credential expression* is a logical expression over credentials with constraints on their attributes. A credential expression serves to denote the combinations of credentials that satisfy it. We call those combinations the *solutions* of the expression. For the purpose of trust negotiation, credential expressions can 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 entity. A credential expression can also be used as a *policy* governing access to a resource. Access to the resource is granted to an entity when a solution is presented that consists of one or more chains ending in credentials owned by the entity. The resource is *unlocked* by the solution. Except where otherwise noted, in this paper, a policy is a credential expression.

A policy is *mobile* if it is sent from one entity to another as part of automatic or semiautomatic trust establishment. Mobile policies have been used in prior systems to express requirements a client must meet to obtain service. When a client makes a service request without sufficient credentials attached to authorize service, the policy governing that service is returned. A policy communicated in this way is viewed as a request for credentials that would unlock the resource it governs. A mobile policy enables the client to select in private 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. In these systems, 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

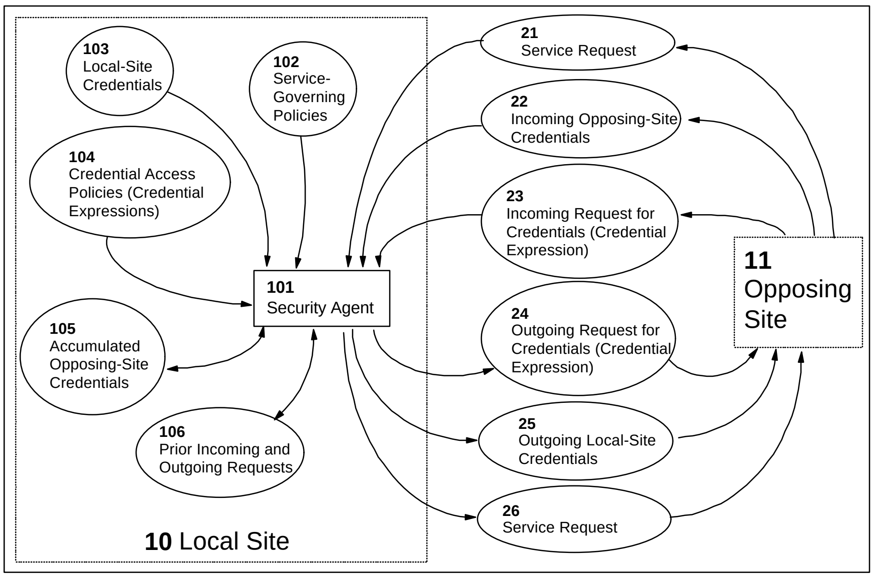
Suppose a client wishes to do business with a new service, but is willing to disclose sensitive credentials only after a minimal degree of trust has been established. Current credential systems do not address credential sensitivity. When sensitive credentials are considered, the decision to disclose a sensitive credential to a new service is left up to a user at the client. More specifically, client-credential submission policies[1] 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 and does not address how the user decides to trust a service.

Winslett et al. recognized the potential to use server credentials to establish client trust. They present an approach for clients to present credentials to servers and note the relevance of such machinery for the reverse scenario, in which servers woo clients by presenting their own credentials. In [Winslett97] each service establishes a policy that is sent to the client by a security agent representing the server when the client requests that service. The client analyzes the server's policy to determine which credentials are needed in support of the client request. In the reverse, a client may request credentials from a server prior to doing business with the server.

This reversal forms a good basis for establishing client trust required for application-level transactions. However, it is inadequate for encouraging clients to disclose their credentials. Suppose the client established a policy identifying credentials required from the server prior to the client disclosing any credentials. This policy would be sent to the server as a counter request when the client received from the server a credential request, such as a service-governing policy. It would then be useless for the server to request client credentials before disclosing its own

[1] This policy is not a credential expression.

credentials, as doing so would introduce a cyclic dependence and deadlock. The problem with



103 Local-Site Credentials
102 Service-Governing Policies
104 Credential Access Policies (Credential Expressions)
101 Security Agent
105 Accumulated Opposing-Site Credentials
106 Prior Incoming and Outgoing Requests
10 Local Site
21 Service Request
22 Incoming Opposing-Site Credentials
23 Incoming Request for Credentials (Credential Expression)
24 Outgoing Request for Credentials (Credential Expression)
25 Outgoing Local-Site Credentials
26 Service Request
11 Opposing Site

Figure 2. The role of the security agent. The local site represents either client or server. When it represents the client, objects 21 and 102 are not manifested. When it represents a server, object 26 is not manifested. When it represents a stateless server, objects 26, 105, and 106 are not manifested.

this model is that all client credentials are governed by the same policy and, hence, any request by the server for client credentials leads to an identical counter request from the client.

4 The Trust Negotiation Framework

The Trust Negotiation Framework solves the problem of establishing trust sufficient to allow disclosure of the credentials that must be exchanged to authorize application-level transactions. It supports assigning different access policies to different credentials, and issuing counter requests that combine the access policies of credentials that solve the incoming request. It also supports constructing counter requests in a way that takes advantage of the credentials already available.

The framework consists of an architecture for supporting fully automated trust negotiation between client and server, together with a partial specification of the behavior of the components of the architecture. The framework leaves the functional specification of certain components up to a *negotiation strategy*, with respect to which the framework is parameterized. Having differing characteristics, different negotiation strategies satisfy different priorities, such as minimizing credential disclosure and negotiating trust successfully whenever possible, as discussed further in Section 5. One strategy characteristic is which credentials to disclose and when. A second is how to formulate a counter-request, if any, when trust has not yet been established. A third is when to give up on a negotiation, without establishing trust. Different negotiation strategies can then be adopted to accommodate different negotiation priorities.

Each of the two negotiation participants is represented in trust negotiations by a *security agent* (SA), as in the simpler negotiations of Ching et al. and Winslett et al. The role of the security agent is illustrated in Figure 2. It depicts the security agent (101) of one site and several contextual factors that the SA considers when managing a negotiation. The local site (10) represents either the client or the server. The SA manages the disclosure of local-site credentials (103). In our framework, each locally-owned credential is governed by an access policy (104), which has the same form as a service-governing policy. The access policy identifies opposing-site credentials that would unlock disclosure of the local-site credential to that opposing site.

Our framework assumes that servers are stateless, although it could be adapted to take advantage of stateful servers. The client (not shown in Figure 2) initiates the trust negotiation by making a service request. Ching et al. and Winslett et al. provide a method whereby the client SA intercepts the client's service request and relays it to the server SA that manages service authorization. The service (not shown in Figure 2) is accessible only via this SA. Upon receiving a request for service (21), the server SA makes an appropriate authorization decision based on a service-governing policy (102). When the client SA is familiar with the service and its governing policy, it can attach appropriate credentials (25) to the service request so that the service will be authorized. The server SA determines whether the policy governing the requested service is satisfied by such credentials, if any. If the policy is satisfied, the service is authorized and the request is forwarded to the server, which provides the service to the client. The trust negotiation has completed successfully. Otherwise, the server SA sends the service-governing policy[2] to the client SA as a request for credentials (23). The client SA can then select a combination of credentials that satisfies the policy, and attach those credentials (25) to a repetition of the original service request (26), which it previously stored for this purpose.

As discussed in Section 3, above, prior trust systems did not enable the client SA to make independent trust decisions about providing the requested credentials to an unfamiliar server. Our framework enables each SA to decide automatically, based on pre-established credential access policies and without user intervention, which credentials to disclose (25). If the client SA cannot satisfy the server's request for credentials by using credentials whose access polices are unlocked, it introduces further stages to the trust negotiation. These stages seek to build mutual trust through credential exchange. In each stage, the active entity can respond to a request for credentials by providing credentials, by formulating a counter-request (24) for opposing-site credentials, or both. The client SA, being stateful, can also repeat a previous request for credentials that has not yet been satisfied. By exchanging credentials and requests for credentials, the two entities endeavor to establish trust required to authorize service. Eventually, either the client SA abandons the attempt to negotiate trust, or the negotiation succeeds. The negotiation succeeds when the client SA can satisfy the original service-governing policy by using client credentials whose access policies are satisfied by available opposing site credentials (22 and 105). In this case, the client SA repeats the original service request (26) with adequate credentials attached (25) to authorize service.

4.1 Functional Specification of the Framework

The following is a functional specification of the framework. The specification of functions in the third group below, CounterToServiceGoverningPolicy, CounterToServiceGoverningPolicy, and HandleServerResponseToCredRequest, are determined by the negotiation strategy chosen to instantiate the framework.

The first three expression-evaluation utilities are used to determine if credentials can be disclosed.

Satisfied (expression, credentials) determines whether a credential expression is satisfied by credentials in a given set. If so, it returns a solution. It returns the empty set if no solution exists.

Unlocked (localCredentials, remoteCredentials) is used to determine local credentials that can be disclosed. It takes a set of local credentials and a set of remote credentials and returns the subset of the local credentials whose access control policies are satisfied by the remote credentials.

Locked (localCredentials, remoteCredentials) is similar to Unlocked, but returns the subset of the local credentials whose access control policies are *not* satisfied by the remote credentials.

The next five functions are called to end a negotiation stage.

ReturnServiceResultToClient (serviceRequest) is used by the server SA when service is authorized. It invokes the desired service, which then returns the service result to the client.

[2] The server SA could also attach some server credentials at this point, a possibility that we ignore to simplify the presentation.

ReturnServiceGoverningPolicyToClient (policy) is called by the server SA to send a service-governing policy to the client SA.

ExitWithNegotiationFailure () is called by the client SA when it determines that a negotiation has failed. The negotiation terminates and failure is reported by the client SA to the client.

RepeatServiceRequest (clientCredentials) is invoked by the client SA when the negotiation succeeds in unlocking a set of client credentials that satisfy the service governing policy. That set is passed as the function's argument. The function attaches the set of client credentials to the repeated service request, which it sends to the server SA.

SendCounterRequest (request, credentials) is called to send a counter request to the opposing site. It takes an outgoing request for opposing-site credentials and any local credentials to be disclosed. Although the behavior of this function is independent of the negotiation strategy, the form of the request is not.

The next three functions are specified by the negotiation strategy. They are parameters of the framework that must be instantiated to obtain a complete system specification. Although all instances of the framework can use the same type of policy to govern services and credentials, the form of counter requests is, like these functions, determined by the negotiation strategy.

CounterToServiceGoverningPolicy (policy, clientCredentials) is invoked by the client SA when it receives a service governing policy that can be satisfied, but only by using some locked, sensitive credentials. This function determines a counter request for server credentials and sends it to the server SA. It may also send some unlocked client credentials.

HandleRequestForServerCredentials (clientRequest, clientCredentials, serverCredentials) is invoked by the server SA to handle a request from the client SA for server credentials. It takes a client request for server credentials, a set of submitted client credentials, and a set of server credentials. It determines and sends to the client SA a counter request, credentials to disclose, both, or neither.

HandleServerResponseToCredRequest (serverRequest, serverCredentials, clientCredentials) is invoked by the client SA when it gets a response to a request for server credentials. It takes a server request for client credentials, a set of submitted server credentials, either or both of which could be empty, and a set of client credentials. It determines whether the negotiation should be terminated with failure, has succeeded, or should be continued. If the negotiation has succeeded, the function repeats the original service request with appropriate credentials attached. If the negotiation should be continued, a request for credentials is sent to the server SA, possibly with client credentials attached.

The first of the following two functions is invoked by the server SA to handle an incoming request for service. The second is invoked by the client SA when it receives a service governing policy.

HandleRequestForService (serviceRequest, clientCredentials), defined below, is invoked by the server SA to handle an incoming request for service. It takes the service request and any attached credentials. It returns the service result when access is granted and the service governing policy when access is denied.

```
policy = LookupServiceGoverningPolicy (serviceRequest)
creds = Satisfied (policy, clientCredentials)
if (creds) ReturnServiceResultToClient (serviceRequest)
else       ReturnServiceGoverningPolicyToClient (policy)
```

HandleServiceGoverningPolicy (policy, clientCredentials), defined below, is invoked by the client SA when it receives a service governing policy from the server SA in response to a service request. It repeats the service request if the service governing policy has a solution consisting entirely of unlocked client credentials. Otherwise, if there is a solution that includes some locked, sensitive client credentials, the function sends the server SA a counter request for server credentials to unlock them. If the service governing policy cannot be satisfied at all by the client's credentials, the negotiation has failed.

```
creds = Satisfied (policy, Unlocked (clientCredentials, ∅))
if (creds)  RepeatServiceRequest (creds)
else if (Satisfied (policy, clientCredentials))
    CounterToServiceGoverningPolicy (policy, clientCredentials)
else     ExitWithNegotiationFailure()
```

5 Negotiation Strategy

In the Trust Negotiation Framework, the negotiation strategy controls the search for a successful negotiation. The strategy controls which credentials an SA requests for the purpose of unlocking local-site credentials, as well as which credentials to disclose and when. Success occurs when client credentials that satisfy the service-governing policy are unlocked. Success is not always possible. One side or the other may not possess needed credentials, or needed credentials from both sides may be governed by policies that impose cyclic dependencies on them. The strategy determines when the client gives up on a negotiation.

Several desirable strategy properties emerge from these observations. Ideally, a strategy should lead to a successful negotiation when one exists. It should terminate with failure when success is impossible. It should do so without disclosing any credentials that are not essential to a successful negotiation. A strategy should be reasonably efficient.

It remains an open problem to find a strategy that possesses all four of these properties or to show that none exists. We discuss two strategies that each satisfy some, but not all, of the properties. The first, discussed in Section 5.1, discloses no credentials that are not essential to a successful negotiation. When negotiation does not succeed, no credentials are disclosed by either participant. However, this strategy does not guarantee that a successful negotiation will be realized, even when success is possible. Termination and efficiency also present difficulties. Section 5.2 discusses the second negotiation strategy, which has different priorities. It is efficient and it is guaranteed to find a successful negotiation when one exists. However, it discloses credentials eagerly and without concern for the necessity of the disclosure. Both strategies discussed here are formally specified in [WSJ].

5.1 A Parsimonious Strategy

The outline of this strategy is as follows. A request for credentials has the same form as an access policy. When an SA receives a request for credentials, it tries to find a solution among its credentials. If none can be found, the negotiation terminates with failure. If a solution exists, the policies governing the credentials that solve the request are combined to form a counter request for credentials, which is sent to the opposing site. No credentials are disclosed by either side until an entity receives a request it can solve with credentials that are all unprotected. At that point, negotiation success is guaranteed. The unprotected credentials are disclosed. The client SA then replays, backwards, the sequence of requests it made to the server up to this point. The first time the client sent each of those requests, the server responded with a counter request for credentials. When it first received that counter request, the client SA selected a solution. During replay, that solution is unlocked for disclosure to the server. Thus, with each request it replays, the client SA attaches credentials that solve the counter request it received from the server SA the first time it sent that request. A typical scenario is illustrated in Figure 3.

Sometimes this strategy must be forced to terminate, lest the two participants continue making counter requests of one another forever. One approach has the client check, when it computes a solution to the incoming request, whether it has arrived at the same solution (or a subset) at a prior stage in the current negotiation. If so, it halts with failure. Unfortunately, this solution permits a number of negotiation stages that exceeds any polynomial in the number of credentials possessed by the client. A heuristical alternative is also possible: just give up after some fixed number of stages when a solution has not yet been found.

The parsimonious strategy discussed here is representative of a family of strategies that have two characteristics. First, at each SA, when an incoming request for credentials is received, the outgoing counter request, sent in response, is sufficient to ensure that any solution to it would unlock credentials satisfying the incoming request. Second, no credentials flow until a request is received that can be solved with unprotected credentials. Strategies in this family share the property that no credential is disclosed except as necessary to achieve negotiation success.

A variant of the parsimonious strategy might be able to preserve this need-to-know property, while finding a successful negotiation whenever one exists. It would need to explore multiple solutions to each incoming request, rather than picking just one. In principle, this could be achieved through backtracking or by basing counter requests on all possible solutions. By using one of these approaches it should be possible to construct a strategy that minimizes credential disclosure and succeeds whenever possible. However, both of these approaches have rather serious efficiency problems. While this discussion shows that parsimonious strategies are possible, it remains to be seen whether they can be made practical.

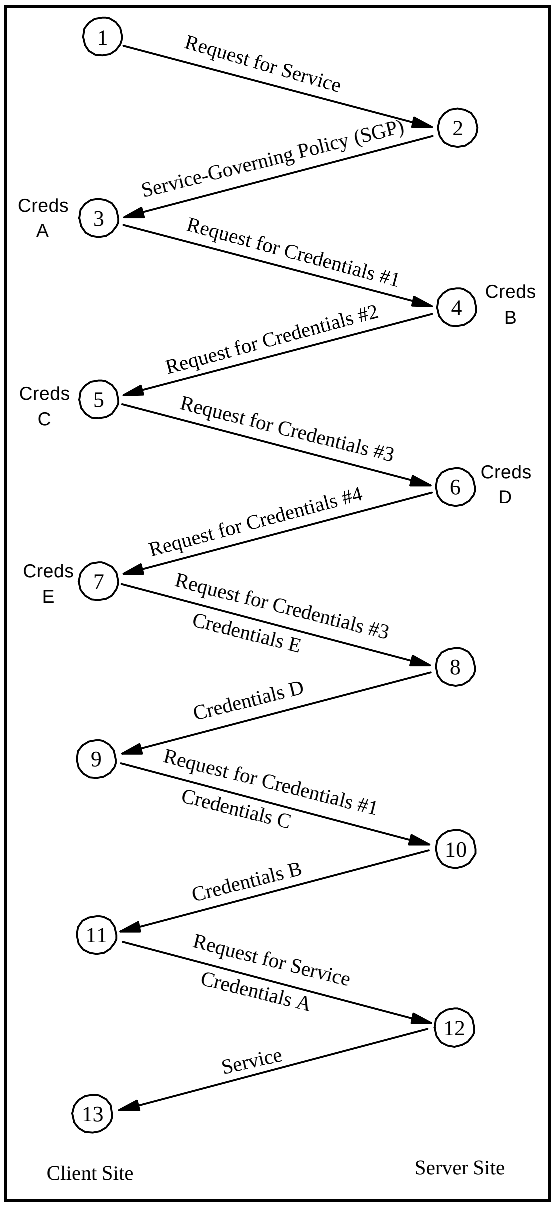

1
Request for Service
2
Service-Governing Policy (SGP)
Creds A
3
Request for Credentials #1
4
Creds B
Request for Credentials #2
Creds C
5
Request for Credentials #3
6
Creds D
Request for Credentials #4
Creds E
7
Request for Credentials #3
Credentials E
8
Credentials D
9
Request for Credentials #1
Credentials C
10
Credentials B
11
Request for Service
Credentials A
12
Service
13
Client Site
Server Site

Figure 3. A typical parsimonious negotiation scenario. The circles represent stages in the negotiation. Arcs are messages labeled with their contents. Stages 3-7 are decorated with credential sets that satisfy the incoming credential requests. Outgoing counter requests are based on the access policies of those credential sets. In later stages, note that the client replays its credential requests, sending the credential sets it found earlier to unlock the credentials it is, again, requesting.

5.2 Eager Negotiation Strategy

The parsimonious strategy discussed in Section 5.1 minimzes credential disclosure, but frequently fails when success is possible. In this section we explore a family of strategies that make the opposite compromise. These *eager* strategies disclose credentials as soon as permitted by their governing policies, and thereby always succeeds when success is possible. They are also efficient.

In the simplest eager negotiation 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. Eventually either the service governing policy is satisfied by unlocked and disclosed credentials and the negotiation succeeds, or no further credentials are unlocked and the negotiation fails. Furthermore, the strategy is reasonably efficient. The number of credential exchanges is bounded by the number of credentials each participant has.

While this simple strategy may find an important role in practice, an eager strategy need not be quite so cavalier about credential disclosure. Requests for credentials can serve to focus the disclosures on credentials that are potentially relevant to unlocking client credentials that satisfy the service-governing policy. Although more could be done to reduce unnecessary disclosures, the strategy presented in [WSJ] seems to point the way toward one suitable for wider use.

For an eager strategy to focus credential exchange on relevant credentials, credential requests must also be exchanged. As in the parsimonious strategy, outgoing counter requests are based on the policies governing credentials requested by the incoming request that they counter. However, in an eager strategy, counter requests must request all credentials that are potentially relevant to a successful negotiation. We call such a request an *eager* request to distinguish it from the focused, precise requests used in the parsimonious negotiation strategy above. The details of representing an eager request are omitted here. However, two general requirements can be identified.

First, when a request for credentials arrives, the counter request sent in response must express *necessary* requirements for unlocking requested credentials. That is, any opposing site credential that is potentially helpful in unlocking local credentials that satisfy the incoming request must satisfy the outgoing counter request. In effect, this says that all relevant credentials are requested. This is necessary to ensure that negotiations succeed whenever possible.

Second, an eager request is satisfied by a single credential. It is used to request all credentials that satisfy it. However, the credentials that satisfy it do so independently of one another. This requirement makes efficient derivation of counter requests straightforward.

Notice that both of these requirements are met in the simple strategy sketched above, where the implicit request for credentials say, “give me each of your credentials.” Also notice that the second requirement is inconsistent with the counter request expressing *sufficient* requirements for unlocking requested credentials. That is, opposing site credentials that satisfy the counter request are not generally sufficient to unlock local credentials that satisfy the original incoming request.

In any eager negotiation strategy, two sets are associated with each participant: (1) the set of credentials it possesses that the opposing site has requested and (2) the set of credentials it has disclosed to the opposing site. Each of these sets grows monotonically with the advancing stages of the negotiation. Because an eager strategy requests credentials as soon as they become relevant and discloses them as soon as they become unlocked, if a stage occurs in which neither set grows, the negotiation has failed and can be terminated. It follows that the number of negotiation steps is linear in the number of credentials possessed by the client.

6 Related Work

Credential-based authentication and authorization systems can be divided into three groups: identity-based, property-based, and capability-based systems. Originally, public key certificates, such as X.509 [x509a, x509b] and PGP [pgp], simply bound keys to names (although X.509 version 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. Property-based credentials -- the credentials discussed in this paper -- 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 entities without requiring familiarity with the actual entities. Systems have emerged recently that use property-based credentials to manage trust in decentralized, distributed systems [Winslett97, Seamons97, Johnston98, and HRLa]. Capability-based systems [Blaze99, SPKI], 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 represent authorization of specific actions with the application server.

Winslett et al. focus on establishing trust with no prior knowledge between client and server. 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. 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 line of research is the only prior work that addresses credential sensitivity. It does this by using mobile policies to support private client selection of credentials to submit for authorization.

Johnston et al. use attribute certificates (property-based credentials) and use-condition certificates (policy assertions) together 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 in order to determine if all the use conditions are met. The certificates are assumed not to be sensitive.

The Trust Establishment Project at the IBM Haifa Research Laboratory [HRLa, HRLb] has developed a system for establishing trust between strangers according to policies that specify constraints on the contents of public-key certificates. They assume credentials are not sensitive and freely available. 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. The collector feature could be adopted by security agents in our work. Their Trust Policy Language (TPL) is an XML-based language designed to map a certificate holder to a role based on the subject's certificates, a given role-assignment policy set by the owner of a resource, and on the roles of the issuers of the certificates. The role can be supplied to an existing role-based access control mechanism. TPL maps the issuers of all supporting credentials to a role during authentication. An example negotiation presented in the appendix uses a simplified language based on aspects of TPL and makes a few subtle, yet significant, additions.

The capability-based KeyNote system of Blaze et al. is designed to manage 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, which 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 polices 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 without knowledge of that application. The IETF Simple Public Key Infrastructure [SPKI] uses a similar approach to that of KeyNote by embedding authorizations directly in certificates.

SSL [SSL], the predominant credential-exchange mechanism in use on the web today, and its successor TLS [TLSa, TLSb], 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 can be a serious problem when the client and server are strangers. In that case, it is unlikely that any single issuer would be an acceptable authority on all server attributes of interest to all potential clients.

7 Conclusions and Further Work

We have presented a Trust Negotiation Framework for incrementally establishing mutual trust. The framework solves the problem of establishing trust sufficient to allow disclosure of sensitive credentials that must be exchanged to authorize application-level operations. The framework is parameterized by a negotiation strategy, which must be specified to instantiate the framework. We have discussed two negotiation strategies, one parsimonious, the other eager. These strategies are formally specified in [WSJ].

The parsimonious strategy successfully minimizes credential disclosure, but can fail to negotiate successfully when success is possible. It can trivially be made to terminate efficiently without affecting this characterization. However, it can be expected to fail unnecessarily so frequently as to make its practical utility questionable. It seems likely that the strategy could be extended to significantly reduce needless failure. Security agents could respond with counter requests that are sufficient to unlock satisfaction of the incoming request, but that could unlock a variety of solutions to that incoming request. In principle, by considering how to unlock every possible solution to the incoming request, such a approach could construct counter requests that are both necessary and sufficient to unlock satisfaction of the incoming request. However, it seems highly unlikely that any such method could be made efficient.

The eager strategy discussed here seems more practical. It negotiates efficiently, finding any negotiation solution that exists. However, it does not minimize credential disclosure. It remains open whether any efficient negotiation strategy can ensure that negotiation succeeds whenever possible while disclosing only credentials that are necessary for success. One approach is to pick one of the two properties as a requirement and approximate the other heuristically.

Alternatively, it seems practical and preferable to make the selection between these two strategies at a much finer granularity. Credential access policies could 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 access policy. One flagged "parsimonious" would be disclosed only as part of some minimal exchange leading to a successful negotiation. A hybrid negotiation strategy would begin 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 eager phase will succeed. If not, phase two of the hybrid strategy would use a parsimonious strategy to attempt to establish trust by using all the credentials. Phase two would take advantage of credentials exchanged during phase one. In such a hybrid negotiation, it would be necessary for the client to make the determination when phase one has completed unsuccessfully and phase two begins. The client would indicate in each request it makes to the server which strategy it is currently employing, eager or parsimonious. The server would then respond accordingly.

The strategies we discuss assume that both participants cooperate in using the same strategy. Further research is required to determine whether and how that assumption can be relaxed. In particular, a parsimonious negotiation guarantees minimal credential disclosure only when both parties "bargain in good faith." It assumes that if a security agent responds to in incoming request by issuing an outgoing counter request, then if the security agent receives opposing-site credentials that satisfy the counter request, together with a repetition of the original incoming request, it will return local-site credentials that satisfy that original request. Thus each negotiation participant requires a certain degree of trust that the other will bargain in good faith. One advantage of a hybrid negotiation strategy, such as the one sketched above, could be that the eager phase could seek to establish this trust required for parsimonious negotiation. Other solutions are also possible and should be considered.

The framework presented here addresses the need of the client (or its security agent) for trust that enables it to disclose credentials to the server (or its security agent). It does not address the need of the client for trust that enables it to request service from the server. The framework can assist in negotiating the server's disclosure of credentials that would establish that trust. However, it does not 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.

8 References

[Bina94] 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, September 1994.

[Blaze99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The KeyNote Trust Management System," work in progress, Internet Draft, March 1999.

[Blaze98] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis, "KeyNote: Trust Management for Public-Key Infrastructures," Cambridge 1998 Security Protocols International Workshop, England, 1998.

[Blaze96] Matt Blaze, Joan Feigenbaum, and Jack Lacy, "Decentralized Trust Management," *1996 IEEE Conference on Privacy and Security*, Oakland, 1996.

[Ching96] 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.

[Johnston98] William Johnston, Srilekha Mudumbai, and Mary 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*.

[HRLa] Yosi Mass, Amir Herzberg, Dalit Naor, Yiftach Ravid, and Joris 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.

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

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

[Schneier96] Bruce Schneier, Applied Cryptography, John Wiley and Sons, Inc., second edition, 1996.

[Seamons97] 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 12, 1997. Available at http://clement.info.umoncton.ca/~lpnet/proceedings97/

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

[SSL] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol," Netscape Communications Corporation, Nov. 18, 1996.

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

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

[WSJ]	W. Winsborough, K. Seamons, V. Jones, “Automated Trust Negotiation: Managing Disclosure of Sensitive Credentials,” Transarc Research Group White Paper, May 1999.
[Winslett97]	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.
[x509a]	International Telegraph and Telephone Consultative Committee (CCITT). The Directory --- Authentication Framework, Recommendation X.509 1993 update.
[x509b]	Information Technology --- Open Systems Interconnection --- The Directory: Authentication Framework, Recommendation X.509, ISO-IEC 9594-8.

9 Acknowledgments

This research was funded by AFRL award F30602-98-C-0222 via North Carolina State University subcontract 98-0973-02.