Receipt-Mode Trust Negotiation: Efficient Authorization Through Outsourced Interactions

Andrew K. Adams[†‡], Adam J. Lee[‡], Daniel Mossé[‡]
† Pittsburgh Supercomputing Center
‡ Department of Computer Science, University of Pittsburgh
akadams@psc.edu, {adamlee, mosse}@cs.pitt.edu

ABSTRACT

In trust negotiation approaches to authorization, previously unacquainted entities establish trust in one another gradually via the bilateral and iterative exchange of policies and digital credentials. Although this affords resource providers with an expressive means of access control for open systems, the trust negotiation process incurs non-trivial computational and communications costs. In this paper, we propose *Receipt-Mode Trust Negotiation* (RMTN) as a means of mitigating the performance penalties on servers that use trust negotiation. RMTN provides a means of off-loading the majority of the trust negotiation process to delegated receipt-generating *helper servers*. RMTN ensures that helpers produce correct trust negotiation protocol receipts, and that the helpers are incapable of impersonating the resource server outside of the RMTN protocol. We describe an initial implementation of our RMTN protocol on a Linux testbed, discuss the security of this protocol, and present experimental results indicating that the receipt-mode protocol does indeed enhance the performance of resource servers that rely on trust negotiation approaches to authorization.

Keywords

Security, optimization, load balancing, trust negotiation

1. INTRODUCTION

In open computing systems where resources are shared across organizational boundaries, users and resource providers may not always have trust relationships that are established a priori. As such, traditional identity-based authorization mechanisms are undesirable. In trust negotiation approaches to authorization (e.g., [7, 15, 17, 20]), access control policies are expressed as declarative specifications of the *attributes* that must be possessed by a principal in order to be granted access to a particular resource. Attributes are encoded in unforgeable and verifiable digital credentials issued by third party certifiers like businesses, professional organizations, or governmental bodies. Since sensitive credentials may themselves be protected by access control policies of their own, a trust negotiation session is a bilateral and iterative exchange of policies and credentials with the end goal of establishing trust between previously unacquainted principals.

Although trust negotiation offers an attractive and expressive means of access control for open system environments, its utility comes at the cost of non-trivial computational and communication overheads. In particular, servers that rely on trust negotiation approaches to access control must not only serve the content that they provide, but also manage the state for all ongoing negotiation sessions. Processing many concurrent trust negotiation sessions can quickly overload such a server and can prevent it from carrying out its primary task, namely serving content to authorized users. The standard solution to this type of problem is load-balancing: replicating a service over additional *similar* hardware and distributing the workload across these additional servers. Unfortunately, naively replicating security systems gives rise to a variety of complications. For instance, are access control policies duplicated across all servers? If so, how are they kept consistent over time? Is the server's credential replicated? If so, how can identity uniqueness still be maintained? When a principal is authorized by one server, is her proof of authorization transferable to all servers? If so, how is this authorization synchronized?

A very constrained version of load-balancing can be employed in which a principal is *locked* to the same server for authorization and resource access for *every* request (i.e., load-balancing keyed on principal id and resource). However, this defeats the inherent qualities of pooled server resources: redundancy and optimal workload distribution. Furthermore, it makes little sense to replicate the core functionality of a resource server to compensate for overheads induced solely by its authorization subsystem. Ideally, it would be desirable to replicate only the security functionality that is actually creating the performance bottleneck, and to do so in a fashion that mitigates potential avenues of attack enabled by the replication of security infrastructure.

Receipt-mode Trust Negotiation (RMTN) pursues exactly this strategy for enhancing the performance of the access-control system. The objective of RMTN is to reduce the server's access control overheads by off-loading the expensive portions of the trust negotiation process to remote *helper servers* (also called helper nodes or simply helpers), while providing high assurance that the remotely negotiated trust was performed correctly. That is, RMTN allows the server to focus on serving content and simply verify that the trust negotiation process was carried out correctly and securely. Additionally, RMTN allows the off-loaded trust negotiation processes to be load-balanced (in any manner) to further improve the performance of the access-control system. Finally, RMTN leverages economics in load-balancing, as the hardware to process trust negotiation does not necessarily have to be comparable to the hardware required to serve resource requests. In exploring the RMTN

process, we make the following contributions:

- We present a RMTN mechanism that reduces the server-side overheads of the trust negotiation process, thereby allowing servers to focus solely on serving resource requests.
- We show that our protocol is secure. This entails showing that two properties are respected: the server can verify that the trust negotiation process carried out by a delegated helper node was in fact executed correctly, and that helper nodes cannot impersonate the resource server in any context other than executing the RMTN protocol, including being hosted in a hostile environment.
- We demonstrate that our Linux-based prototype RMTN system does, in fact, allow servers to safely and securely serve content in a scalable manner while leveraging trust negotiation to authorize client resource requests.

The rest of this paper is organized as follows. In Section 2, we briefly explain the problems associated with naive implementations of trust negotiation, and present a set of desirable qualities for systems designed to load-balance trust negotiation sessions. Section 3 discusses the principals within the system, and 4 presents the Receipt-Mode Trust Negotiation protocol and its high-level properties. In Section 5, we describe the performance of an implementation of RMTN for the World Wide Web. We then discuss related work in Section 6 and present our conclusions and directions for future work in Section 7.

2. SYSTEM REQUIREMENTS

Trust negotiation is expensive: state must be kept between *all* participating principals, the process is sequential, and trust negotiation requires extensive asymmetric cryptographic operations (e.g., digital signatures and signature verifications). Not surprisingly, off-loading trust negotiation can spare a resource server from having to expend cycles during trust negotiation. However, distributing the security components of a system presents new challenges. For instance, how do we ensure that the off-loaded trust negotiation was performed correctly? Likewise, how do we ensure that delegated components cannot maliciously masquerade as the server? Can the system scale in a manner consistent with typical load-balancing schemes? Is it too burdensome to implement? At a minimum, our new system must exhibit the following properties:

- **Correctness.** An off-loaded security system must be able to produce the same access control decisions that the non-enhanced system produces when using the same policy. That is, for all principals, the access control decision generated must be consistent regardless of whether an access control is made locally or off-loaded to another component. If such a system is unable to produce *correct* results, resources could be compromised.
- **Security.** An off-loaded security system should be no less secure than the original system. Current trust negotiation mechanisms provide non-repudiation and prevent against certificate misuse (both fraud and privacy). Failing to provide these same *security* guarantees is unacceptable.
- **Efficiency.** An off-loaded security system should be manageable, but more specifically, such a system needs to be more efficient (economically and with resources) than naive load-balancing/replication and easy to integrate with existing services. Failing to provide an *efficient* system will result in non-deployment.
- **Scalability.** An off-loaded security system should exhibit performance characteristics that scale at least as well as canonical load-balancing mechanisms (i.e., full replication). If such a system fails to be *scalable*, resources will be wasted.

This is the minimum set of desired properties for off-loaded security systems, as the particular application domain can always add further specialized requirements.

3. SYSTEM ARCHITECTURE

In this section, we first describe the players in our proposed protocol and then discuss the nuance associated with delegating the control of trust negotiation sessions to one or more helper servers.

3.1 Principals and Assumptions

In an open system, several principals can interact in an effort to consume resources that are protected by authorization policies governing their access. The following describes the participants in our proposed RMTN protocols, including their job functions and any assumptions regarding their trust.

Client A *client* seeks access to a policy-protected resource through the use of applications under its control. The client is capable of entering into a trust negotiation session in an attempt to satisfy the access control policies protecting the resource. The client's *trust negotiation agent* (a process executing at the trust negotiation) has access to the client's attribute certificates and private keys.

Server A *server* is responsible for serving content or resources that may be protected by access control policies. The server's trust negotiation agent has access to the policies protecting its resources, as well as the certificates and private keys describing the attributes of the server. The server does not inherently trust clients, and will grant client access to a particular resource if and only if the client can present a set of credentials that satisfy the policy protecting the requested resource (via trust negotiation).

Helper Server A *helper server* is installed by a server admin that wishes to off-load its trust negotiation process. In particular, the helper remotely replaces the software trust negotiation agent that would otherwise run on the server. The helper has access to a copy of the policies that the server uses to protect its resources, and the helper operates using restricted versions of the server's credentials (see Section 3.2).

3.2 Provisioning Helpers

The notion of using a helper server to facilitate trust negotiation session may, at first blush, appear to violate basic security principles. In particular, helper servers require access to *every* credential used by the server, yet conventional wisdom says that replicating secrets leads to a higher likelihood that these secrets will be compromised. And indeed, if the server's policies and credentials are simply replicated, this would be the case: any compromised or malicious helper could impersonate the server. This is clearly problematic. To address this issue, we borrow the concept of proxy certificates [18] from the grid computing literature.

Proxy certificates were originally developed to allow a user's certificates and keys to be managed by third parties during the execution of long-running processes. Rather than requiring a user to surrender control of her long-term identity certificate, she can instead create a temporary (proxy certificate, private key) pair. The limited-lifetime proxy certificate is bound to the long term certificate, and indicates that the owner of the proxy certificate has the

ability to act on behalf of the owner of the long-term certificate. An optional policy field can be used to indicate any restrictions on exactly how the owner of the proxy certificate "speaks for" the owner of the long-term certificate.

In our approach, we assume that helper servers are provisioned with proxy credentials bound to the server's long-term attribute credentials, rather than the server's sensitive attribute credentials themselves. Further, a critical X.509 extension can be used to indicate that these proxy certificates are to be used by RMTN servers only. Since this extension is critical, nodes that do not understand how to interpret the extension should reject the credential outright, as defined in the X.509 specification [1]. Nodes that do understand this extension will only allow the proxy certificate to be used in response to a RMTN request. This use of proxy certificates is critical to our ability to safely provision helper nodes without introducing security vulnerabilities into the system (e.g., malicious helpers compromising the long-term credentials of the server).

4. PROTOCOL OVERVIEW

In this section, we provide an overview of our proposed RMTN protocol. We begin with a discussion of each RMTN message type, describe the basic protocol, and then present a performance analysis of the client and server effort.

4.1 Protocol Philosophy and Basic Messages

Several system implementations for providing TN services exist today (e.g., TrustBuilder2 [13], Trust-X [7], PP-Trust-X [17], and PeerTrust [15]). It is desirable for the RMTN protocol to leverage existing TN applications and, therefore, conscious decision was made to ensure that the RMTN protocol is as agnostic as possible to the TN messages generated by the application libraries. To this end, our RMTN protocols are designed to facilitate and orchestrate control transfers between the application-level protocols running on some server to one or more distributed trust negotiation agents. As such, the actual trust negotiation protocol implemented by the agents can be left entirely unmodified, and the RMTN protocol requires that only two new message types be specified:

RMTN Redirect (RMTN-RDR) This message allows an application-level protocol running on some server to inform a remote client that a transfer of control to some remote trust negotiation helper is necessary.

RMTN Receipt (RMTN-RCP) This message allows the RMTN subsystem to transfer control from the RMTN handler application back to the application level protocol running on the original remote server.

In short, the information contained in an RMTN-RDR message enables the client to contact an appropriate helper (or helpers) for the purpose of authorization via trust negotiation, and allows the remote helper to determine which resource protection policy it should use in the resulting trust negotiation session. The RMTN-RCP message is used to encapsulate the state of the resulting trust negotiation, and allows the server to validate how and why the client became authorized to access its service.

4.2 Basic RMTN

The basic form of our proposed RMTN protocol is illustrated in Figure 1. This protocol begins when a client requests a policy-protected resource from a server (message A). The server, recognizing that the request is for a policy-protected resource, issues a RMTN-RDR message to the client that designates (at least) one remote helper that the client should use in order to off-load the hard

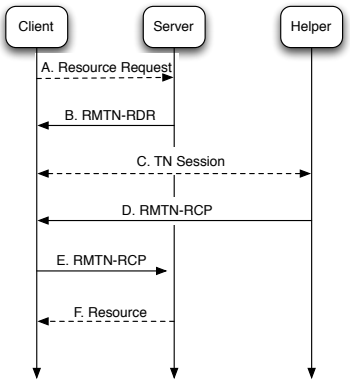

Figure 1: Basic RMTN protocol.

work from the server (message B). Based on the parameters in the RMTN-RDR message, the client and helper enter into a trust negotiation session in order to find a set of the client's credentials that satisfy the resource access policy (messages C). Assuming a satisfying set of credentials for the policy can be found, the helper server generates a RMTN-RCP message for the client that encapsulates a digitally-signed, minimal representation of the protocol state (message D).

The client can then present this receipt to the server (message E). After verifying that the signature on the receipt was created by an authorized helper and that the credentials presented by the client match those identified in the receipt, the server can verify that the credentials satisfy the policy implied by the client's resource request and serve the resource (message F). This is accomplished by invoking a policy compliance checker on the policy protecting the resource and the set of credentials provided by the client. This step allows the server to "check the work" of the helper for correctness *without* requiring the server to execute the entire protocol.

Client/Server Delay Trade-Offs. The response time gains should be immediately realized in RMTN provided that the time to perform TN is much greater than the time to serve the resource originally requested. The time on the server to process a resource request is $O(Stn + Sr)$, where $O(Stn)$ is the time to negotiate trust with the client and $O(Sr)$ is the time to serve the resource. The addition of a helper changes the response time to $O(Srdr + Htn + Crcp + Sr)$, where $O(Srdr)$ is the time to send the client the RMTN-RDR message, $O(Htn)$ is the time to negotiate trust on the helper and $O(Crcp)$ is the time the client takes in transmitting the receipt to the server. However, $O(Htn)$ occurs on the helper, so local processing resources on the server are only $O(Srdr + Crcp + Sr)$. And if we assume delivery time of messages is negligible compared to TN or serving resources, then the server is characterized by $O(Sr)$.

Obviously, the response time the client sees is still limited by $O(Htn + Sr)$, but if the server has sufficient work, and the time to serve a resource is much less than the time to negotiate trust, i.e., $O(Sr) \ll O(Stn)$, then the significant amount of time *not* spent in TN on the server can be used to process more requests—the aggregation of clients reap the improved response time. This expected load-balancing advantage is validated in Section 5.

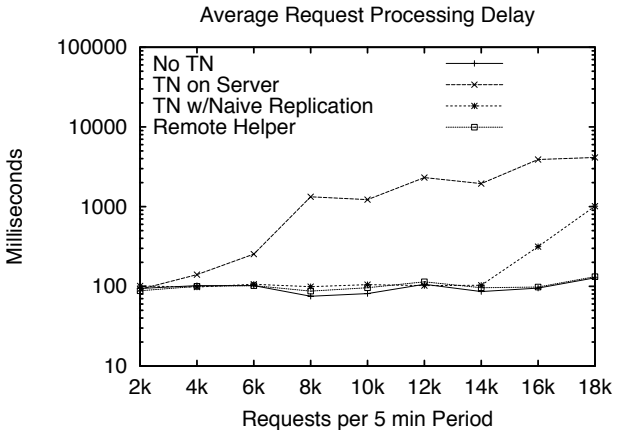

Figure 2: Average request delay per webpage download with and without trust negotiation enabled (log scale y-axis).

A Note on Initial Configuration. Before either of the above RMTN protocols can be used, the helper nodes deployed in the system require certain configuration information. Specifically, each helper node needs (i) an identity certificate and signing key, (ii) proxy certificates for each of the server attribute certificates that it will have access to, and (iii) copies of the server policies that it will be relied upon to enforce. While configuring the long-lived identity certificates for a helper node should happen in an offline manner, the remainder of the configuration can take place over a private and authenticated channel (e.g., a TLS tunnel [9]) in an on-demand manner. This type of on-the-fly provisioning is typical in the grid environments for which proxy certificates were originally designed, and does not conflict with our security goals.

5. EXPERIMENTAL EVALUATION

Our RMTN evaluation infrastructure is comprised of five AMD Opteron 1352 (1GHz) processor-based servers, and a single P5-based host running Linux 2.6.9. One AMD server, running Apache httpd 2.0.52, acts as the resource server; two AMD servers act as traffic generators; two AMD servers are used as RMTN helpers; and the lower-end host acts as a simple client process used to measure system response times. We generate traffic to our web server that ramps up from 4k requests to 18k requests (average over 5 minute intervals). Our web server has HTML files of $\approx 300KB$, up to $\approx 700KB$. One page is protected by an *extremely* simple TrustBuilder2 policy (the policy requires nine credentials to be exchanged within two rounds, i.e., three TrustBuilder2 messages, during trust negotiation prior to termination). A client session consists of one access to a protected resource, and nine subsequent accesses to unprotected resources.

To validate the performance of RMTN, four configurations were tested measuring *average response time* (see Figure 2 and note the log scale Y axis). "No TN" represents a baseline *without* trust negotiation, in which the response time is about 100ms, regardless of the load (i.e., the server was *not* the bottleneck). "TN on Server" is when the server itself executes trust negotiation causing delays of up to two orders of magnitude on *every* page access (i.e., not just the policy-protected page). "TN w/Naive Replication" initially shows improvement, however, as the workload increases naive replication begins to suffer similar to the "TN on Server", not surprisingly. Finally, when trust negotiation is off-loaded to a remote helper node ("Remote Helper") shows the intuition behind RMTN: allowing the server to focus on serving content can ensure that content is served efficiently. In particular, RMTN allows server throughput to remain comparable to the baseline case (i.e., "No TN") even when serving thousands of clients per minute.

6. RELATED WORK

Trust negotiation has been an active area of research within the security community over the last several years. In addition to research on foundational issues like languages for expressing resource access policies (e.g., [5, 6, 14]) and logics for reasoning about the outcomes of negotiations (e.g., [8,22]), a number of research groups have developed trust negotiation implementations for the purposes of experimentation [7, 13, 21]. Others have also looked at optimizations to the trust negotiation process, though these optimizations are typically related to either the amount of information revealed during or the efficiency of a *single* trust negotiation session [7, 12, 16, 23]. In contrast, we focus on optimizing the way in which a server handles *many* concurrent trust negotiation sessions.

In many ways, receipt-mode trust negotiation can be classified as a *load balancing* mechanism. Several techniques for load balancing appear in the domain of networking (e.g., [2, 3, 10]). In these solutions, packets or network flows are assigned to a pool of resources to avoid overloading any particular resource, increasing the reliability and throughput. To the best of our knowledge, load balancing has not been previously applied within the realm of trust negotiation. Naive replication of server-side credentials increases the likelihood of compromising the server's long-term secrets; our RMTN protocols avoid this problem by replicating short-lived proxy certificates that allow helper nodes to operate on the server's behalf without risking the safety of long-term secrets.

The crux of the RMTN protocol is its ability to allow servers *safely* delegate the execution of a trust negotiation session to a set of helper nodes. Outsourcing of security protocols has been considered within the context of cryptographic puzzles (e.g., [4, 11]) that force clients to carry out an expensive computation prior to being granted access to a server. To be effective, puzzles need to be inexpensive to generate and verify, yet hard to solve. Waters et al. [19] show how puzzle generation can be offloaded to a so-called bastion to reduce the computational burden at servers. Although similar in spirit to RMTN, these puzzles are used to rate-limit requests rather than authenticate users.

7. CONCLUSIONS AND FUTURE WORK

Trust negotiation has been proposed as an expressive means of access control for open environments in which users and resources may not have a priori established relationships. Despite its advantages, the state management overheads associated with conducting many trust negotiation sessions can be significant. In this paper, we proposed remote-mode trust negotiation (RMTN) as a means of mitigating these costs. RMTN *safely* delegates the interactive portions of the trust negotiation process to helper nodes that carry out the trust negotiation process and generate third-party verifiable receipts attesting to the outcomes of the processes. As a result, resource servers need only to verify this receipt prior to serving content, which is a lightweight process. We proposed the RMTN protocol in this paper, experimentally verified that RMTN greatly decreases server response time, and showed that the use of the protocol compromises neither the correctness of the trust negotiation process nor the long-term credentials held by the resource server.

In the future, we hope to extend this work in two directions. First, it would be beneficial to carry out experiments that test the

utility of inexpensive platforms for RMTN. As the main drawback of trust negotiation is its serial nature, rather than computational or communication overheads, it seems reasonable to expect that RMTN helpers can be much less powerful than resource servers. The ability to leverage old or outdated hardware to respond dynamically to the changing access needs of an organization would solidify RMTN's utility over that of naive replication of resource servers. Second, protocols that enable the resource server to synchronize policies and proxy certificates with its helpers in an efficient manner need to be researched to allow servers to keep their security environment consistent.

Acknowledgements. This research was supported in part by the National Science Foundation under awards CCF–0916015, CNS–1017229, CNS–0964295, CCF–0811295, and CNS–1012070.

8. REFERENCES

[1] Information technology - open systems interconnection - the directory: Public-key and attribute certificate frameworks, March 2000.

[2] Link aggregation (ieee 802.1ax), 2008. `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4668665`,.

[3] Y. Amir, R. Caudy, A. Munjal, T. Schlossnagle, and C. Tutu. N-way fail-over infrastructure for reliable servers and routers. In *DSN*, pages 403–, 2003.

[4] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant authentication with client puzzles. *Cambridge Security Protocols Workshop 2000*, Apr. 2000.

[5] M. Y. Becker and P. Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *5th IEEE International Workshop on Policies for Distributed Systems and Networks*, 2004.

[6] E. Bertino, E. Ferrari, and A. C. Squicciarini. X -TNL: An XML-based language for trust negotiations. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '03)*, 2003.

[7] E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-x: A peer-to-peer framework for trust establishment. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):827–842, 2004.

[8] P. Bonatti and P. Samarati. Regulating service access and information release on the web. In *7th ACM Conference on Computer and Communications Security*, pages 134–143, 2000.

[9] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, Aug. 2008.

[10] C. C. Fan. The raincore distributed session service for networking elements.

[11] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. *Proceedings of NDSS '99 (Networks and Distributed Security Systems)*, pages 151–165, 1999.

[12] A. J. Lee and M. Winslett. Towards and efficient and language-agnostic compliance checker for trust negotiation systems. In *3rd ACM Symposium on Information, Computer, and Communication Security (ASIACCS '08)*, Mar. 2008.

[13] A. J. Lee, M. Winslett, and K. J. Perano. Trustbuilder2: A reconfigurable framework for trust negotiation. In *Third IFIP WG 11.11 International Conference on Trust Management (IFIPTM 2009)*, June 2009.

[14] N. Li and J. Mitchell. RT: A role-based trust-management framework. In *Third DARPA Information Survivability Conference and Exposition*, Apr. 2003.

[15] W. Nejdl, D. Olmedilla, and M. Winslett. Peertrust: Automated trust negotiation for peers on the semantic web. In *LDB Workshop on Secure Data Management (SDM)*, volume 3178 of *Lecture Notes in Computer Science*, pages 118–132, 2004.

[16] T. Ryutov, L. Zhou, C. Neuman, T. Leithead, and K. E. Seamons. Adaptive trust negotiation and access control. In *10th ACM Symposium on Access Control Models and Technologies*, June 2005.

[17] A. Squicciarini, E. Bertino, E. Ferrari, F. Paci, and B. Thuraisingham. Pp-trust-x: A system for privacy preserving trust negotiations, 2007.

[18] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820 (Proposed Standard), June 2004.

[19] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for DoS resistance. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 246–256, Oct. 2004.

[20] W. H. Winsborough and N. Li. Automated trust negotiation. In *In DARPA Information Survivability Conference and Exposition, volume I*, pages 88–102. IEEE Press, 2000.

[21] M. Winslett, T. Yu, K. E. Seamons, A. Hess, J. Jacobson, R. Jarvis, B. Smith, and L. Yu. Negotiating trust on the web. *IEEE Internet Computing*, 6(6):30–37, Nov./Dec. 2002.

[22] M. Winslett, C. Zhang, and P. A. Bonatti. PeerAccess: A logic for distributed authorization. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS 2005)*, Nov. 2005.

[23] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies in automated trust negotiation. *ACM Transaction on Information and System Security (TISSEC)*, pages 1–42, February 2003.