STRONGMAN: A SCALABLE SOLUTION TO TRUST MANAGEMENT IN NETWORKS

Angelos Dennis Keromytis

A Dissertation

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial

Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2001

Acknowledgements

There are several people I owe a note of thanks to, for their role in my graduate studies:

Jonathan Smith, for being my advisor and a good friend. I cannot imagine going through graduate school without Jonathan as my advisor, despite of (or perhaps because of) all the grief I gave him through the years and which he endured heroically, while striving to make me understand the intricacies of research and academic life.

John Ioannidis, for convincing me to pursue graduate studies in the first place, and for his collaboration and friendship throughout the years (and the many sushi expeditions).

Matt Blaze, for pointing out to me some of the interesting problems in security (which eventually led to this thesis), and for being my mentor (with John Ioannidis) at AT&T Labs Research.

Joan Feigenbaum, for the insightful conversations we had over the years, and for always being punctual. It is always a pleasure to work with someone who insists on not deferring tasks (such as finishing a paper) until the last minute!

Dave Farber, for sharing with me his perspective on security (and computer science), formed through his many years of presence in the field.

Michael Greenwald, for giving me extremely detailed comments on this thesis and asking the tough questions. Michael's comments led to the development of the scalability model discussed in Section 5.5.5.1.

Sotiris Ioannidis, for being an excellent collaborator (Sotiris was instrumental in the development of the distributed firewall, discussed in Chapter 6), and for being a good friend and colleague for over 10 years.

Scott Alexander, for being there and helping me understand what this “research” thing was all about when I was beginning my studies. Scott has been a good friend through the years.

Bill Arbaugh, for the many interesting conversations and his (many) nutty ideas. Working with Bill (in SANE, DHCP++, and AEGIS) has been an educational experience (in a good way!).

Vassilis Prevelakis, for sharing with me his experience and ideas during the last year of my work. Vassilis also came up with the scalability example in Section 1.1.

Stefan Mitchev, for instrumenting the experiment discussed in Section 5.5.

Mitch Marcus, for serving as my thesis committee chair.

Michael Felker, the department graduate administrator. Mike makes the department tick.

The fine folks at the business office (especially Mark and Gail), for putting up with me all these years.

Finally, my parents (Hi mom!) for everything else.

Abstract

STRONGMAN: A Scalable Solution to Trust Management in Networks

Angelos Dennis Keromytis

Jonathan M. Smith

The design principle of restricting local autonomy only where necessary for global robustness has led to a scalable Internet. Unfortunately, this scalability and capacity for distributed control has not been achieved in the mechanisms for specifying and enforcing access-control policies. With the increasing size and complexity of networks, management of security is becoming a more serious problem. In particular, as the complexity of a network increases (measured in terms of number of users, protocols, applications, network elements, topological constraints, and functionality expectations), both the management and the enforcement mechanisms fail to provide the necessary flexibility, manageability, and performance.

This dissertation addresses the problem of scalable, high-performance distributed access control in very large, multi-application networks. The proposed architecture, named STRONGMAN, demonstrates three new approaches to providing efficient local policy enforcement complying with global security policies. First is the use of local compliance checkers at the enforcement points to provide great local autonomy within the constraints of a global security policy. Second is the ability to compose policy rules into a coherent and enforceable set. These policies may potentially span network layer and application boundaries. Third is the "lazy binding" of policies to reduce resource consumption; this on-demand binding scales because there is significant spatial and temporal locality of reference that can be exploited in access control decisions.

To demonstrate the value and feasibility of STRONGMAN, I designed and implemented an access control management system based on the KeyNote trust-management system. The experiments contacted using the resulting system show that the underlying network's performance (in terms of throughput and latency) is largely unaffected, and in some cases improved, by utilizing the concepts of distributed access control and "lazy binding".

As a result, STRONGMAN is the *first* fully automated access control management system that can scale, both in terms of management and in terms of performance, to arbitrarily large networks. The two-layer approach to management allows for a detachment of the high levels of the system, which mainly impact its manageability, from the lower levels, which affect performance and scalability.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Technology trends in recent years have resulted in the rapid deployment of interconnected, non-research computer networks. The growth rate of the Internet has been impressive [Wiz, Tec], and there are indications that this will continue to be the case for some time yet.

One particular reason for this trend has been the "network effect," *i.e.,* the fact that the usefulness of some types of technologies increases with the number of users of that technology. In the case of computer networks, increasing numbers of online services attract increasing numbers of users (including corporate entities), a fact which in turn drives more information and services to become available online. Thus, administrators have to deal with large numbers of network-attached devices and users, and a variety of protocols and mechanisms[1]. As some of these resources are not public, (*e.g.,* financial, military, personal data), various protection mechanisms have been deployed to mediate access to them. Figure 1.1 shows some of the mechanisms developed to counter threats at different layers in the network stack.

[1]An indication of the number of new services and protocols being deployed can be found in the number of new Request For Comment documents that have been issued the past few years[Edi]: 1992 - 92 RFCs, 1993 - 173, 1994 - 184, 1995 - 130, 1996 - 171, 1997 - 190, 1998 - 235, 1999 - 260, 2000 - 278. Not all of these documents refer to distinct protocols or services, but they extend or modify existing protocols in some way.

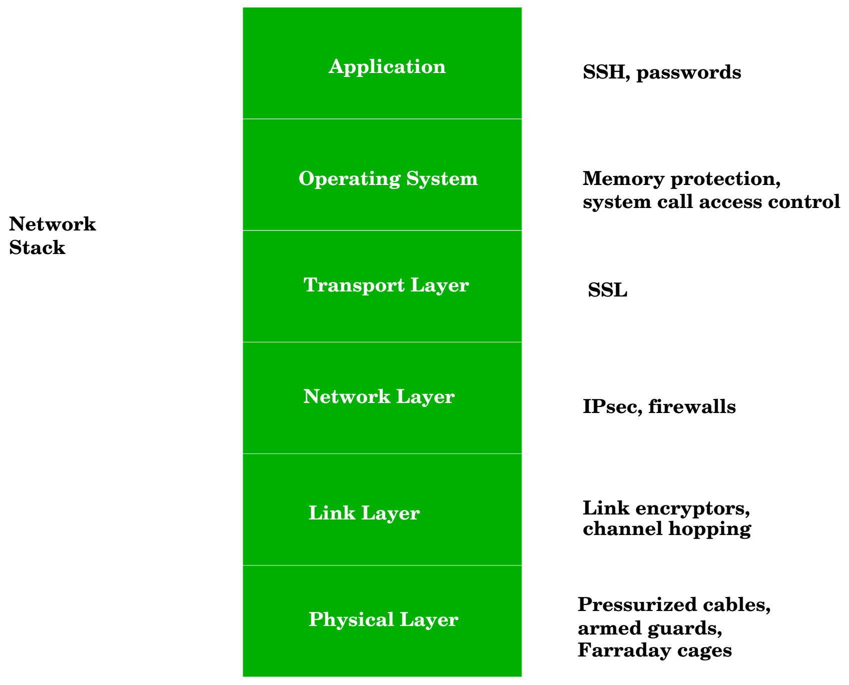

Application
SSH, passwords
Operating System
Memory protection, system call access control
Network Stack
Transport Layer
SSL
Network Layer
IPsec, firewalls
Link Layer
Link encryptors, channel hopping
Physical Layer
Pressurized cables, armed guards, Farraday cages

Figure 1.1: Security mechanisms at various layers of the network stack (note that this is not the "standard" OSI stack).

Even networks without direct attachments to the Internet (such as military tactical networks) are of such size and complexity that they cannot depend on physical security measures to prevent unauthorized access. Since the same types of equipment and protocols/applications are often used in these "private" networks, the same, or very similar, security mechanisms are employed.

The variety of security mechanisms can be explained by the differences in the threat models they address, as well as the guarantees they offer: IP firewalls offer a convenient, from an administrator's point of view, place to perform access control on packets and connections due to the restrictions they impose on the network topology, as seen in Figure 1.2. Firewalls do not enforce any end-to-end security properties by themselves; they are systems dedicated to examining network traffic between a protected network and the rest of the world. A firewall can permit or deny a particular packet (or connection) to pass through it, based on the access control policy the administrator defined and the information contained on the packet[2], but it cannot directly protect the traffic from eavesdropping or modification once it has traversed the firewall. On the other hand, network-layer encryption offers end-to-end secrecy and integrity guarantees, but does not directly address the issue of access control.

Furthermore, these same networks and devices are increasingly used in more versatile ways; examples of such usage scenarios include the "intranets" and "extranets", where parts of an otherwise protected network are exposed to another entity for the purposes of collaboration (business or otherwise), and tele-commuting.

One particular side-effect of these new usage models has been the need for access control mechanisms that can operate throughout a network, and enforce a coherent security policy. To return to the example of using a central firewall to protect a network, such an approach does not work very well in the new network environment:

- The assumption that all insiders are trusted [3] has not been valid for a long time. Specific individuals or remote networks may be allowed access to all or parts of the protected infrastructure (extranets, tele-commuting, *etc.*); email-based worms and viruses manage to infect networks behind a firewall because of some internal user accessing their email account from inside the protected network using a protocol that the firewall does not (or cannot) filter. Consequently, the traditional notion of a security perimeter can no longer hold unmodified; for example, it is desirable that tele-commuters' systems comply with the corporate security policy.

[2]Network addresses and ports, and other application-specific information that can be extracted from the packet.

[3]This is an artifact of firewall deployment: internal traffic that is not seen by the firewall cannot be filtered; as a result, internal users can mount attacks on other users and networks without the firewall being able to intervene.

- It is easy for anyone to establish a new, unauthorized entry point to the network without the administrator's knowledge and consent. Various forms of tunnels, wireless, and dial-up access methods allow individuals to establish backdoor access that bypasses all the security mechanisms provided by traditional firewalls. While firewalls are in general not intended to guard against misbehavior by insiders, there is a tension between internal needs for more connectivity and the difficulty of satisfying such needs with a centralized firewall.

- There are protocols that are difficult to process at the firewall because the latter lacks certain knowledge that is readily available at the endpoints (such as what traffic flows are part of the same application/communication session); FTP and RealAudio are two such protocols. Although there exist application-level proxies that handle such protocols, such solutions are viewed as architecturally "unclean" and in some cases too invasive.

- End-to-end encryption can also be a threat to firewalls [Bel99], as it prevents them from looking at the packet fields necessary to do filtering. Allowing end-to-end encryption through a firewall implies considerable trust in the users by the administrators.

- Finally, there is an increasing need for finer-grained (and even application-specific) access control which standard firewalls cannot readily accommodate without greatly increasing their complexity and processing requirements. For example, fine-grained access on URLs served by an internal web server to roaming users is easiest done on the web server itself; a firewall would have to perform application layer filtering (or interpose itself as a proxy), and bypass any end-to-end encryption security.

Thus, it appears that access control must become an end-to-end consideration, similar to authentication and confidentiality. This is perhaps not very surprising, as the IP architecture itself was based on the end-to-end design principle [SRC84, Cla88]. Simply stated, this means that communication properties that must hold end-to-end should be provided by mechanisms at the end points. Seen from a different point of view, the end point (in terms of access control) must move from the firewall to the end-host, when a network has to exhibit a high degree of decentralized access control.

To manage access control in these networks and deliver the required services, tools and architectures that allow the administrators to effectively cope with the increased scale and complexity of the network entities (devices, users, protocols, security policy enforcement points) and the policies that define the allowed interactions between these. In particular, the primary method of addressing scalability issues in networking (and other areas) has been replication at the system and administrator level. One way of achieving the latter is through a “separation of duty” structure, where different administrators manage different aspects of the network’s operations.

Current tools either ignore, or do not sufficiently address separation of duty concerns, as we shall see in Section 2. Even in small networks, administrators have trouble handling the configuration of a small number of firewalls [Woo01]. The results of this can be seen in studies of network intrusions and their causes [How97]: an increasing number of vulnerabilities can be directly attributed to misconfiguration, with an even larger percentage of intrusions indirectly caused by administration failures.

The situation is equally bad in the case of mechanism scalability: most access control mechanisms become a bottleneck in the overall system performance as the level of replication increases, which is an attempt to meet the increased demand in bandwidth (disk or network) and processing. To better illustrate this, let us consider a simple example.

1.1 Access Control Scalability

Imagine a building with N doors. People wishing to enter the building show up at one of the door; all doors are equivalent for the purpose of accessing the building.

In a simple configuration, each door has a guard that examines the person's identification (authentication) and checks the list of people that are allowed to enter the building (access control). If the person is on the list, he is allowed in the building.

To scale for many visitors, we have to increase the number of doors. In the case of the traditional access control (the guards), we have the problem of distributing the list to all the guards and maintaining that list. Furthermore, if the number of potential visitors is large, the list becomes very large and the guards have to spend time and effort looking up people in that list (and lifting the list!). Although we have multiple doors, and we can hire many guards, the work of the guards increases rapidly with the number of users, because that work depends on the size of the list.

In the STRONGMAN case, the guards are replaced with locks on the doors. Each person has a key and that key grants access to the building. Let us assume momentarily that all visitors have the same key (access control policy); in that case, any visitor can enter through any door. The work in performing an access control decision does not depend on the number of doors. Also since each visitor is supplying the key, the complexity of the locks on each door is independent of the total number of visitors or the number of other doors. As the complexity of the mechanism increases (more sophisticated locks, taking more time to operate) the throughput per door may go down, but this can be fixed by adding more doors.

The problem of giving all the visitors the same key is also addressed in STRONGMAN via the delegation of authority, as we shall see in Chapter 4.

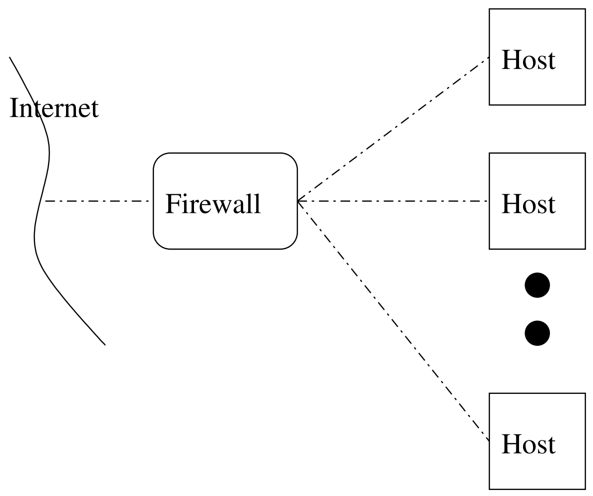


Figure 1.2: A firewall's bottleneck topology.

1.2 Requirements

From the discussion above, it is clear that access control management systems have several requirements:

- The system must be able to support the security policy requirements of many diverse applications, given the large number in use today. (Throughout this thesis, the term "applications" is used to mean services and protocols that require access control configuration. These applications can be security-oriented, *e.g.,* a network layer security protocol, or they may be consumers of security services, *e.g.,* a web server.)

- The increasing size and complexity of networks strains the ability of administrators to effectively manage their systems. The traditional way of handling scale at the human level has been decentralization of management and delegation of authority. This approach is evident throughout the complete range of human activities (*i.e.,* most, if not all, effective large "systems" involve the creation and maintenance of an administration service where responsibility for different aspects of the system is handled by different entities). Thus, an access control management system for large networks must be able to adapt to different management structures (web of trust, hierarchical management, *etc.*).

- The system should be agnostic with respect to the configuration front-end that administrators use. The first reason for this is to allow a decoupling of the management mechanism, which could potentially be used for the whole lifetime of the network, from the method used to configure it, which may change as a result of new developments in Human-Computer Interaction interfaces, or because of a change in administrators. Secondly, such a system, by allowing the use of different management front-ends for configuring different applications' access control policies, encourages the development and use of front-ends (GUIs, languages, *etc.*) that are tailored to the specific application and its particular nuances.

  We should note that this requirement is not typical for access control management systems; most such systems promote the use of a single configuration front-end for all the applications in the system. Although more research is needed in this area, one can see the parallels between the all-encompassing languages developed in the 1970s and the more recent trend on "domain-specific" languages (languages specifically designed to address a limited application domain, *e.g.,* [HKM+98]).

- The system must be able to handle large numbers of users, applications, and policy evaluation and enforcement points. As we saw above, corporate (and other) networks are rapidly increasing in size; furthermore, new protocols are being deployed (without necessarily deprecating old ones); finally, these same networks are used in increasingly more complicated ways (intranets, extranets, *etc.*).

- A corollary of the above is that the system should be able to handle the common operations (like adding or removing users) efficiently. This is important because, over the lifetime of the system, these overheads will dominate other costs like initial deployment.

- Last but not least, the system must be efficient. It should not impose significant overheads on existing protocols and mechanisms; it should strive to match the performance curve attained by service replication. Ideally, it should even improve performance by addressing any inefficiencies in existing management systems.

1.3 Novelty

In [Bel99, IKBS00], the concept of the Distributed Firewall was introduced, which offers high flexibility in terms of fine-grained network access control. While it is generally agreed that this approach represents a trend in the evolution of network security (as evidenced by the number of "personal firewall"[4] commercial products), there has been some amount of skepticism as to the feasibility of managing a large network composed of such elements (interspersed perhaps with "traditional" firewalls as well). This dissertation focuses on solving the problem of managing the security mechanisms of a large network.

The primary focus of this work is the development of an architecture for managing access control policy in networks with large numbers of users (in the hundreds of thousands or even millions), services and devices, which have a requirement for fine-grain access control. These

[4]Very briefly, this is a firewall that is oriented for the small office or home environment, requiring little or no configuration and offering some basic filtering capabilities. In the same category are software packages that run on a user workstation or laptop and perform the same functionality.

goals are achieved through the careful application of three principles:

- The use of compliance checkers local to the enforcement points to provide great local autonomy within the constraints of a global security policy, also allowing us to enforce policy at an arbitrarily fine granularity.
- The ability to compose policy rules into a coherent enforceable set, potentially across network layer and application boundaries, allowing us to address manageability and scalability concerns.
- The "lazy binding" of policies to reduce resource consumption in storing and processing access control information, by taking advantage of the locality of reference phenomenon in communications, which allows scalability of the security enforcement mechanisms.

1.4 Contributions

The following items are the principal contributions of this dissertation:

- The *first* system that demonstrates **improved** performance as a result of decentralization of access control functions, compared to traditional access control enforcement mechanisms.
- The *first* application of the *trust management* model of authorization on a large-scale system, and an examination of its strengths and weaknesses. We identify the potential architectural bottlenecks and quantify their scaling characteristics with a set of experiments in a controlled environment.
- The development of a scalable (over the number of users, policies, and enforcement nodes) architecture for managing the access control policy of large-scale networks, by using the

three principles outlined in Section 1.3.

- A study of the performance overhead of the mechanisms used by STRONGMAN, using a prototype implementation in OpenBSD.
- A study of the various tradeoffs involved in access revocation in a decentralized access control architecture like STRONGMAN.

1.5 What is Not Covered in this Dissertation

The term *security policy*, as used in Computer Science, is overloaded. An organization's security policy is composed of such diverse elements as access control, intrusion detection, incident response, backup procedures, and legal measures (this list is not, nor is it intended to be, comprehensive). This dissertation only addresses security policy with respect to access control. Access control is a very important aspect of a system's security, and serves as the basis for all the other mechanisms and procedures an organization may utilize. However, as pointed out in [Sch00], this typically causes administrators and managers to overlook the other aspects of a proper security posture.

1.6 Remainder of this Dissertation

The next chapter discusses related work. Chapter 3 presents a short introduction to several technical areas that will be helpful in understanding the remainder of the dissertation. Chapter 4 describes the STRONGMAN architecture and discusses its main components, while Chapter 5 describes the prototype implementation of STRONGMAN for IPsec and HTTP access control, along with the experimental results obtained from testing. Chapter 6 presents some applications based on or derived from STRONGMAN, and possible future work. Finally, Chapter 7 presents the conclusions drawn from this research.

Chapter 2

Related Work

2.1 Access Control and Security Policy

The work by Lampson [Lam71, Lam74] established the ground rules for access control policy specification by introducing the access control matrix as a useful generalization for modeling access control. A concept derived from the access control matrix that is used in many security system is the Access Control List (ACL); this is a list of $<$ *Subject, Object, Access Rights* $>$ tuples, that collectively encompass the access control policy of the entire system, in terms of users, and services or data to which access must be controlled. The focus in both that work and in Taos [WABL94] is on authentication. The latter depended on a unified policy specification and enforcement framework, although it identified credentials (in the form of digitally signed statements) as a scalable authorization mechanism.

There has been some work in embedding information (called "attributes") beyond simple identity inside X.509 [CCI89] certificates. Taken to the extreme, one could encode constraints similar to those in KeyNote inside X.509 certificates. These could then subsume the role of KeyNote credentials, assuming a more evolved evaluation/verification model was also used.

The principles of separation of duty (SoD) described in [AS99] can be directly applied in our work, by mapping the SoD constraints directly into KeyNote policies and credentials. Similarly, the principles of negotiated disclosure of sensitive policies presented in [SWY01] can be applied in work such as ours to limit the risk of exposing the information encoded in credentials to active attackers.

[Aur99] presents an in-depth discussion of the advantages and disadvantages of credential-based access control. [Jae99] discusses the importance of constraining delegation, a fundamental tenet of our approach.

2.2 Languages and Policy Specification

In [BdVS00] the authors propose an algebra of security policies that allows combination of authorization policies specified in different languages and issued by different authorities. The algebraic primitives presented allow for considerable flexibility in policy combination. As the authors discuss, their algebra can be directly translated to boolean predicates that combine the authorization results of the different policy engines. The main disadvantage of this approach is that it assumes that all policies and (more importantly) all necessary supporting information is available at a single decision point, which is a difficult proposition even within the bounds of an operating system. Our observation here is that in fact the decision made by a policy engine can be cached and reused higher in the stack. Although the authors briefly discuss partial evaluation of composition policies, they do so only in the context of their generation and not on enforcement.

The approach taken in Firmato[BMNW99] is that of use of a “network grouping” language that is customized for each managed firewall at that firewall. The language used is independent of the firewalls and routers used, but is limited to packet filtering. Furthermore, it does not handle delegation, nor was it designed to cover different, interacting application domains (IPsec, web access, *etc.*). Policy updates are equivalent to policy initializations in that they require a reloading of all the rules on the affected enforcement points. Finally, the entire relevant policy rule-set has to be available at an enforcement point, causing scale problems with respect to the number of users, peer nodes, and policy entries.

An architecture similar to Firmato is described in [Hin99]. While the latter attempts to cover additional configuration domains (such as QoS), the main differences with Firmato are in the policy description language and in the way the two systems prune the set of rules that have to be enforced by any particular device. While this is an important issue in both papers, we note that it is an artifact of the respective architectures and their treatment of policies as fixed rules that must always be present in the enforcement points. Other similar work includes [Gut97, Mol95, Sol].

[Kan01] describes the use of *virtual tags* as part of evaluating QoS policy for DSCP. Virtual tags are information related to a packet that is not derived from the packet itself, but was produced as a result of matching another policy rule against that packet. The concept of the virtual tags is very similar in some way to the blackboard idea used in PolicyMaker [BFL96] (see Section 3.4.1).

[CLZ99] describes a language for specifying communication security policies, heavily oriented toward IPsec and IKE. SPSL is based on the Routing Policy Specification Language (RPSL) [ABG$^+$98]. While SPSL offers considerable flexibility in specifying IPsec security policies, it does not address delegation of authority, nor is it easily extensible to accommodate other types of applications.

The IETF organized a Policy working group to develop common standards for the various vendors interested in providing policy-based networking devices. The group is working in the area of policy schemata that can be implemented in LDAP directories. There is similar work done in the Distributed Management Task Force (DMTF).

Adaptive policies and their usefulness are discussed in [CL98]. A number of approaches to supporting these are described, with different tradeoffs between flexibility, security, reliability, and performance.

One benefit of loose coupling of policies for different languages is that we can apply consistency analysis such as that described in [CC97] in the component languages separately. The interaction points between different languages may be considered as invariants for the purposes of the analysis.

[PS99] describes an implementation of RBAC (Role-Based Access Control) with rule hierarchies on the web. The authors use "smart certificates" that encode the role information.

Other work in trust-management systems includes REFEREE [CFL$^+$97], which is the first application of a trust management system to web access control, and Delegation Logic [Li00], which is a language for expressing authorization and delegation based on logic-theoretic semantics of logic programs.

2.3 Access Control Systems

Firewalls [CB94, MRA87, Mog89, Eps99, SSB$^+$95, MWL95, GSSC96, NH98] are a well-studied technique for access control policy enforcement in networked environments. However, traditional work has focused on nodes rather than the networks they protect, and enforcement mechanisms rather than policy coordination.

[Ven92] and [LeF92] describe different approaches to host-based enforcement of security policy. These mechanisms depend on the IP addresses for access control, although they could potentially be extended to support some credential-based policy mechanism, especially if used in conjunction with IPsec.

In the OASIS architecture [HBM98], the designers identify the dependencies between different services and the need to coordinate these. They present a role-based system where each principal may be issued with a name by one service on condition that it has already been issued with some specified name of another service. Their system uses event notification to revoke names when the issuing conditions are no longer satisfied, thus revoking access to services that depended on that name. Each service is responsible for performing its own authentication and policy enforcement. However, credentials in that system are limited to verifying membership to a group or role, thus making it necessary to keep policy closely tied to the objects it applies to. Furthermore, OASIS uses delegation in a very limited scope, thus limiting administrative decentralization.

The work described in [HGPS99] proposes a ticket-based architecture using mediators to coordinate policy between different information enclaves. Policy relevant to an object is retrieved by a central repository by the controlling mediator. Mediators also map foreign principals to local entities, assign local proxies to act as trusted delegates of foreign principals, and perform other authorization-related duties. Coordination policy must be explicitly defined by the security administrator of a system, and is separate from (although is taken in consideration along with) access policy.

The NESTOR architecture [BKRY99] defines a framework for automated configuration of networks and their components. NESTOR uses a set of tools for generating and managing a network topology database. It then translates high-level network configuration directives into device-specific commands through an adaptation layer. Emphasis is given on system consistency across updates; this is achieved through a transactional interface. Policy constraints are described in a Java-like language and are enforced by dedicated manager processes. As the authors note, because of this dependency on managing stations, the architecture does not scale well. It is also not clear how amenable this approach is to hierarchical/decentralized administration [Jae99] and separation-of-duty [DOD85, AS99] concerns, due to its view of the network through a central configuration depository.

The Napoleon system [TOB98, TOP99] defines a layered group-based access control scheme that is in some ways similar to the distributed firewall concept presented in [IKBS00], although it is mostly targeted to RMI environments like CORBA. Policies are compiled to access control lists appropriate for each application (in our case, that would be each end host) and pushed out to them at policy creation or update time.

SnareWork [CS96] is a DCE-based system that can provide transparent security services (including access control) to end-applications, through use of wrapper modules that understand the application-specific protocols. Policies are compiled to ACLs and distributed to the various hosts in the secured network. Connections to protected ports are reported to a local security manager which decides whether to drop, allow, or forward them (using DCE RPC) to a remote host, based on the ACLs.

[BGS92] describes an open, scalable mechanism for enforcing security. It argues for a shift to a more decentralized policy specification and enforcement paradigm, without discussing the specifics of policy expression. It emphasizes the need for delegation as a mechanism to achieve scale and decentralization, but focuses on design of protocols for accomplishing this rather than the more high-level requirements on policy expression.

Also in the context of the IETF, the RAP (RSVP Admission Policy) working group has defined the COPS [BCD$^+$00] protocol, as a standard mechanism for moving policy to the devices. This protocol was developed for use in the context of QoS, but is general enough to be used in other application domains.

RADIUS [RRSW97] and its proposed successor, DIAMETER [CRAG99], are similar in some ways to COPS. They require communication with a policy server, which is supplied with all necessary information and is depended upon to make a policy-based decision. Both protocols are oriented toward providing Accounting, Authentication, and Authorization services for dial-up and roaming users.

A protocol for allowing controlled exposure of resources at the network layer is presented in [Tsu92]. It uses the concept of “visas” for packets, which are similar to capabilities, and are verified at boundary controllers of the collaborating security domains.

[SC98] defines a protocol for dynamically discovering, accessing, and processing security policy information. Hosts and networks belong to security domains, and policy servers are responsible for servicing these domains. The protocol used is similar in some ways to the DNS protocol. This protocol is serving as the basis of the IETF IP Security Policy Working Group.

The usefulness of decentralized access control management has been recognized for specific applications, *e.g.,* see [SP98] for an application of this approach in web access control.

2.4 Kerberos

[MNSS87] is an authentication system that uses a central server and a set of secret key protocols, as shown in Figure 2.1, to authenticate clients and give both a client and an application server a secret key for use in protecting further communications. While enforcement is done in a decentralized manner, the system requires high availability and online security for the server. Furthermore, all network nodes are expected to be have their clocks synchronized (typically within a few seconds of each other). The two most important deficiencies of Kerberos are that it does not implement any kind of authorization (applications are expected to make their own access control decisions, based on information they acquire through other means, *e.g.,* Directory Services, local ACLs, database queries), and it is expensive, in terms of administrative effort, to do cross-realm authentication, as this requires all clients to have complete knowledge of the trust relationships between realms (a Kerberos realm is the collection of systems and users managed by a single administrative entity). Both these deficiencies have been addressed:

- Kerberos *tickets* have been extended to contain some authorization information [WD01], albeit still requiring some type of external lookup to determine access privileges. In principle, constraints such as those expressed in KeyNote could be encoded inside Kerberos tickets, and likewise processed at the enforcement point. This approach requires existing protocols and applications to be modified to understand Kerberos authentication; one advantage of STRONGMAN is that KeyNote credentials can easily be piggy-backed on top of the existing certificate exchange mechanisms used by protocols like IKE [HC98] and TLS/SSL [DA99].

- The issue of cross-realm authentication has been addressed by *cross-realm referrals,* introduced in [TKS01]. These allow knowledge of the trust relationships between realms to be

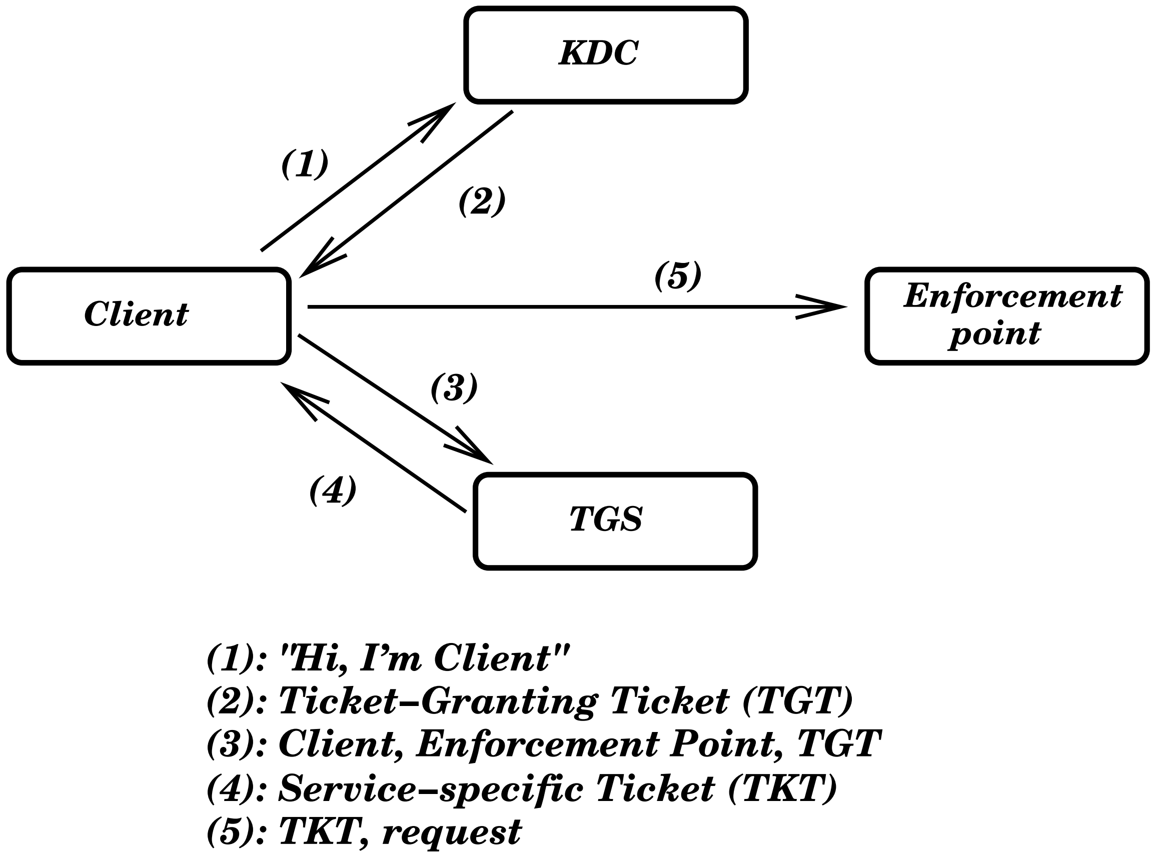

KDC
(1)
(2)
Client
(5)
Enforcement point
(3)
(4)
TGS
(1): "Hi, I'm Client"
(2): Ticket–Granting Ticket (TGT)
(3): Client, Enforcement Point, TGT
(4): Service–specific Ticket (TKT)
(5): TKT, request

Figure 2.1: The Kerberos authentication protocol. Initially, the client authentication to the Key Distribution Center (KDC), which gives it a Ticket Granting Ticket; this step occurs infrequently (typically, once every 8 hours). For each service the client needs to contact, it must then contact the Ticket Granting Service (TGS), which responds with a Ticket (TKT) that is service-specific. The client then contacts the service, providing TKT. Often, the KDC and the TGS are co-located.

moved to the Key Distribution Center (KDC) of each realm. These allow the client to iteratively determine which KDCs it should ask for tickets that construct a trust path between the client's KDC and the application server's KDC, as shown in Figure 2.2. The assumption however is that the next KDC to contact can be determined by examining the name of the service (in particular, the domain name of the host on which the service runs, as found in the DNS database). This makes vertical separation of duty (*e.g.,* sharing management of different applications in a subnet among a group of administrators) very difficult to handle without implementing another layer of access control for the administrators themselves.

Furthermore, since tickets would have to be re-issued fairly often, the secret key for each KDC would have to be online. Thus, compromise of a single KDC would greatly weaken the security of any realms that directly or indirectly trust it.

A more important deficiency, however, lies in the nature of the secret key authentication employed by Kerberos: Referrals can solve the problem of securely determining the identity of the principals and KDCs involved in a request, but they cannot be used to convey hierarchical policy information to the enforcement point, beyond any policy included in the ticket issued by the enforcement point's KDC. While access policy could be encoded in the referrals themselves, these would not be verifiable to the enforcement point (since it does not share a secret key with any of the intermediate KDCs). The intermediate KDCs cannot make an access control decision at the time the referral must be issued, since they do not have any information about the application request itself; even if they did, however, this would be an extremely inefficient approach to access control, since all such KDCs would have to be contacted each time a request is made — with no possibility of ticket and referral caching, as is currently possible. Similar inefficiencies arise when the enforcement point contacts each KDC for every request made by the client. Either of these

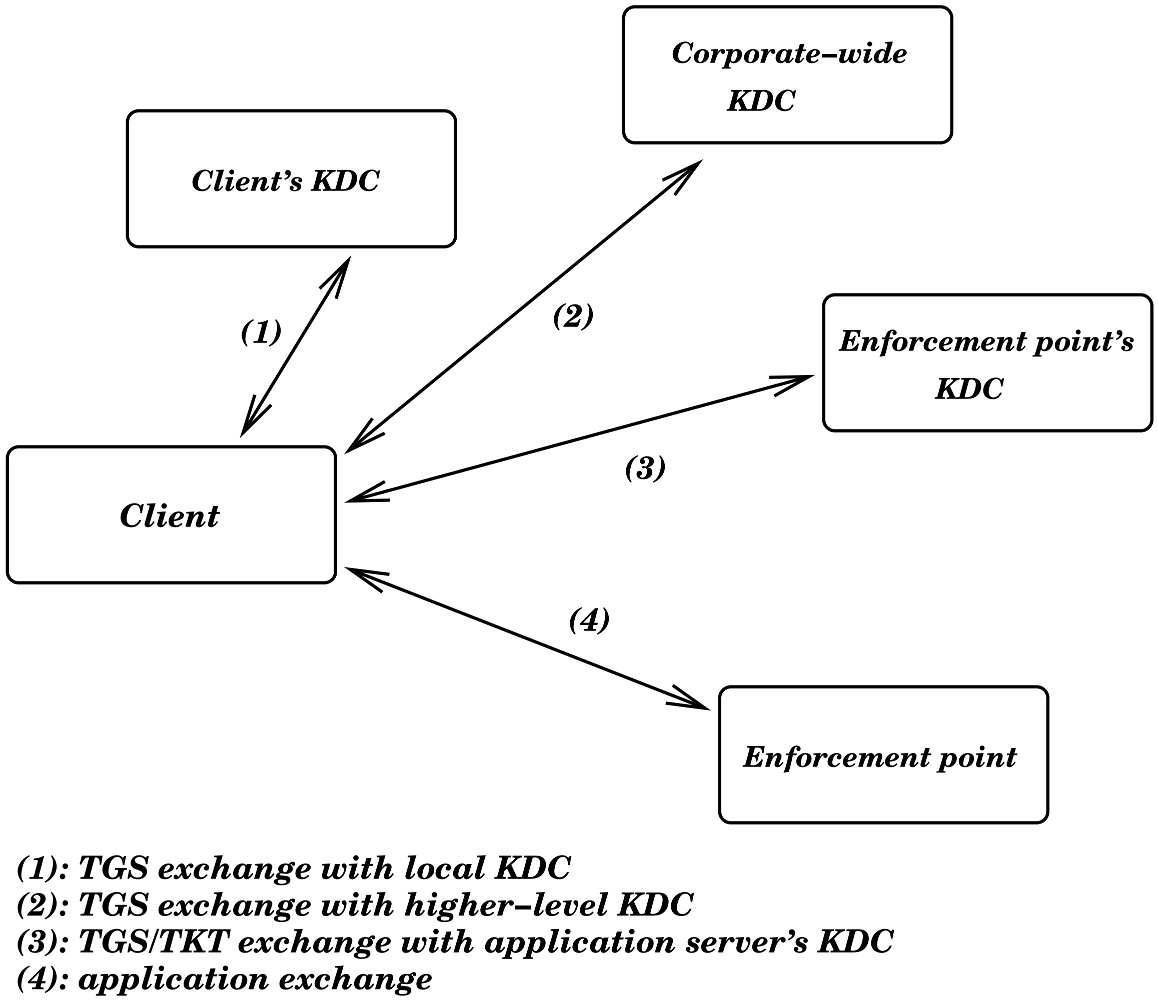

Corporate-wide KDC
Client's KDC
(1)
(2)
Enforcement point's KDC
(3)
Client
(4)
Enforcement point
(1): TGS exchange with local KDC
(2): TGS exchange with higher-level KDC
(3): TGS/TKT exchange with application server's KDC
(4): application exchange

Figure 2.2: Kerberos referrals for cross-realm authentication.

approaches effectively converts a fairly decentralized authentication mechanism into an extremely centralized access control mechanism.

Finally, a referral-based architecture that supports policy dissemination, requires duplication of client information at both the client and the enforcement point's KDC. This is necessary because only the enforcement point's KDC can provide policy information to the enforcement point (encoded inside a ticket), and therefore has to have knowledge of the client's privileges.

2.5 Access Control System Classification

Table 2.1 classifies the various systems based on the requirements we discussed in Section 1.2.

	Multiple Languages	Multiple Applications	Decentralized Management	Scalability	Cheap Updates
[NH98]		**x**			
[Eps99]		**x**			
[HBM98]		**x**			**x**
[Hin99]		**x**			
[Gut97]		**x**			
[BMNW99]					
[Mol95]		**x**			
[Sol]		**x**			
[HGPS99]			**x**		**x**
[BdVS00]	**x**	**x**			
[BKRY99]		**x**			
[Ven92]					
[LeF92]					
[TOB98]		**x**			**x**
[CS96]					
[PS99]					**x**
[CLZ99]					**x**
[BCD$^+$00]					**x**
[RRSW97]					**x**
[BGS92]			**x**		**x**
[MNSS87]		**x**		**x**	**x**
STRONGMAN	**x**	**x**	**x**	**x**	**x**

Table 2.1: System classification.

Chapter 3

Technical Foundations

This section introduces some concepts and systems that will be used throughout this dissertation. The first three sections introduce the concepts of authentication, authorization and access control, and public key cryptography[1] (including digital signatures and public key certificates). The fourth section introduces Trust Management and gives a brief overview of the *PolicyMaker*[BFL96] system. The fifth section discusses *KeyNote*[BFIK99c], which is used as the basis of the STRONGMAN system.

3.1 Authentication

The term authentication in network security and cryptography is used to indicate the process by which one entity convinces another that it has possession of some secret information, for the purpose of "identifying" itself. This identification does not necessarily correspond to a real-world entity; rather, it implies the continuity of a relationship (or, as stated in [Sch00], knowing who to trust and who not to trust).

[1]Section 3.3 is partially based on the terminology used in Arbaugh's dissertation[Arb99].

In computer networks, strong authentication is achieved through cryptographic protocols and algorithms. A discussion of the various constraints and goals in these protocols is beyond the scope of this section. The interested reader is referred to [Sch96].

3.2 Authorization and Access Control

Authorization[2] is the process by which an enforcement point determines whether an entity should be allowed to perform a certain action. Authorization takes place after a principal[3] has been authenticated. Furthermore, authorization occurs within the scope of an access control policy. In simpler terms, the first step in making an access control decision is determining who is making a request; the second step is determining, based on the result of the authentication as well as additional information (the access control policy), whether that request should be allowed.

3.3 Public Key Cryptography

A cryptographic algorithm is "symmetric" if the same key is used to encrypt and decrypt (*e.g.,* DES[NBS77]). Public key systems use two different keys: a private key, $K_{private}$, and a public key, K_{public}, where $D_{K_{public}}(E_{K_{private}}(M)) = M$. Example of public key cryptographic systems are RSA[Lab93] and DSA[NIS94].

Public key cryptographic systems have a significant advantage over symmetric systems in that two principals can exchange a message, or verify the validity of a digital object (see Section 3.3.1), provided they have acquired the peer's public key in some trusted manner. In contrast to symmetric key systems, where the principals must exchange keys in a confidential manner, public keys do not

[2]For the purposes of this thesis, the terms authorization and access control are synonymous.

[3]An entity, such as a user or a process acting on behalf of a user, that can undertake an action in the system.

need to be confidential.

3.3.1 Digital Signatures

Digital signatures use a public key system to provide non-repudiation. To sign an object, the signer computes a function of the object and his private key[4]. The result of this function is verifiable by anyone knowing the corresponding public key. A valid signature assures the verifier that, modulo bad key management practice on the part of the signer or some breakthrough in forgery, the signed object was indeed signed by the signer's private key and that it has not been modified since that time.

3.3.2 Public Key Certificates

Public key certificates are statements made by a principal (as identified by a public key) about another principal (also identified by a public key). Public key certificates are cryptographically signed, such that anyone can verify their integrity (the fact that they have not been modified since the signature was created). Public key certificates are utilized in authentication and authorization protocols because of their natural ability to express delegation (more on this in Section 3.4.1).

A traditional public key certificate cryptographically binds an identity to a public key. In the case of the X.509 standard [CCI89], an identity is represented as a "distinguished name", *e.g.,*

```
/C=US/ST=PA/L=Philadelphia/O=University of Pennsylvania/
OU=Department of Computer and Information Science/CN=
Jonathan M. Smith
```

In more recent public key certificate schemes [EFL$^+$99, BFIK99c] the identity is the public key,

[4]Usually the digest of the object is signed, computed through a cryptographic one-way hash function.

and the binding is between the key and the permissions granted to it. Public key certificates also contain expiration and revocation information.

Revoking a public key certificate means notifying entities that might try to use it that the information contained in it is no longer valid, even though the certificate itself has not expired. Possible reasons for this include theft of the private key used to sign the certificate (in which case, all certificates signed by that key need to be revoked), or discovery that the information contained in the certificate has become inaccurate. There exist various revocation methods (Certificate Revocation Lists (CRLs), Delta-CRLs, Online Certificate Status Protocol (OCSP), refresher certificates), each with its own tradeoffs in terms of the amount of data that needs to be kept around and transmitted, any online availability requirements, and the window of vulnerability.

3.4 Trust Management

A traditional "system-security approach" to the processing of a signed request for action (such as access to a controlled resource) treats the task as a combination of *authentication* and *authorization*. The receiving system first determines *who* signed the request and then queries an internal database to decide *whether* the signer should be granted access to the resources needed to perform the requested action. It has been argued that this is the wrong approach for today's dynamic, internetworked world[BFL96, BFIK99a, Ell99, EFL$^+$99]. In a large, heterogeneous, distributed system, there is a huge set of people (and other entities) who may make requests, as well as a huge set of requests that may be made. These sets change often and cannot be known in advance. Even if the question "who signed this request ?" could be answered reliably, it would not help in deciding whether or not to take the requested action if the requester is someone or something from whom the recipient is hearing for the first time.

The right question in a far-flung, rapidly changing network becomes “is the key that signed this request *authorized* to take this action ?” Traditional name-key mappings and pre-computed access-control matrices are inadequate. The former because they do not convey any access control information, the latter because of the amount of state required: given N users, M objects to which access needs to be restricted, X variables which need to be considered when making an access control decision. we would need access control lists of minimum size $N \times X$ associated with each object, for a total of $N \times M$ policy rules of size X in our system. As the conditions under which access is allowed or denied become more refined (and thus larger), these products increase. In typical systems, the number of users and objects (services) is large, whereas the number of variables is small; however, the combinations of variables in expressing access control policy can be non-trivial (and arbitrarily large, in the worst case). Furthermore, these rules have to be maintained, securely distributed, and stored across the entire network. Thus, one needs a more flexible, more “distributed” approach to authorization.

The *trust-management approach*, initiated by Blaze *et al.* [BFL96], frames the question as follows: “Does the set C of *credentials* prove that the *request r complies* with the local security *policy P* ?” This difference is shown graphically in Figure 3.1.

Each entity that receives requests must have a policy that serves as the ultimate source of authority in the local environment. The entity’s policy may directly authorize certain keys to take certain actions, but more typically it will *delegate* this responsibility to credential issuers that it trusts to have the required domain expertise as well as relationships with potential requesters. The *trust-management engine* is a separate system component that takes (r, C, P) as input, outputs a decision about whether compliance with the policy has been proven, and may also output some additional information about how to proceed if the required proof has not been achieved. Figure 3.2

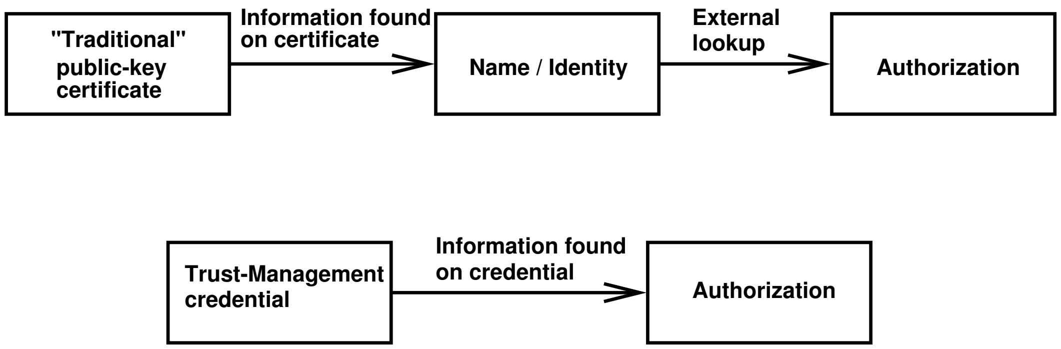

"Traditional" public-key certificate
Information found on certificate
Name / Identity
External lookup
Authorization
Trust-Management credential
Information found on credential
Authorization

Figure 3.1: The difference between access control using traditional public-key certificates and trust management.

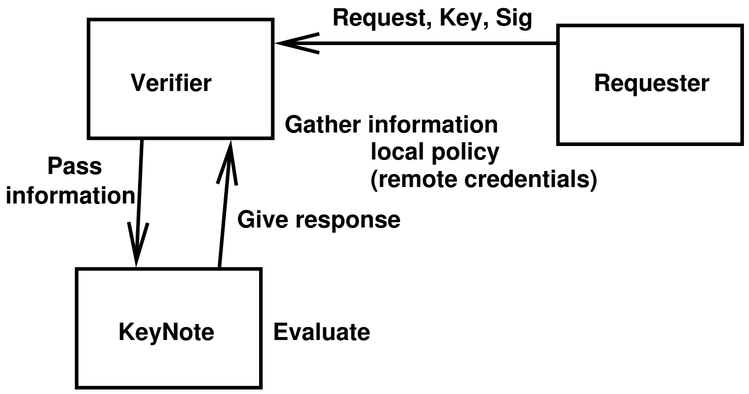


Figure 3.2: Interaction between an application and a trust-management system.

shows an example of the interactions between an application and a trust-management system.

An essential part of the trust-management approach is the use of a *general-purpose, application independent* algorithm for checking proofs of compliance. Why is this a good idea ? Any product or service that requires some form of proof that requested transactions comply with policies, could use a special-purpose algorithm or language implemented from scratch. Such algorithms/languages could be made more expressive and tuned to the particular intricacies of the application. Compared to this, the trust-management approach offers two main advantages.

The first is simply one of engineering: it is preferable (in terms of simplicity and code reuse) to have a “standard” library or module, and a consistent API, that can be used in a variety of different applications.

The second, and perhaps most important gain is in soundness and reliability of both the definition and the implementation of “proof of compliance.” Developers who set out to implement a “simple,” special-purpose compliance checker (in order to avoid what they think are the overly “complicated” syntax and semantics of a universal “meta-policy”) discover that they have underestimated their application’s need for proof and expressiveness. As they discover the full extent of their requirements, they may ultimately wind up implementing a system that is as general and expressive as the “complicated” one they set out to avoid. A general-purpose compliance checker can be explained, formalized, proven correct, and implemented in a standard package, and applications that use it can be assured that the answer returned for any given input (r, C, P) depends only on the input and *not* on any implicit policy idecisions (or bugs) in the design or implementation of the compliance checker.

Basic questions that must be answered in the design of a trust-management engine include:

- How should “proof of compliance” be defined ?
- Should policies and credentials be fully or only partially programmable? In which language or notation should they be expressed ?
- How should responsibility be divided between the trust-management engine and the calling application ? For example, which of these two components should perform the cryptographic signature verification? Should the application fetch all credentials needed for the compliance proof before the trust-management engine is invoked, or may the trust-management engine fetch additional credentials while it is constructing a proof ?

At a high level of abstraction, trust-management systems have five components:

- A language for describing ‘actions’, which are operations with security consequences that are to be controlled by the system.
- A mechanism for identifying ‘principals’, which are entities that can be authorized to perform actions.
- A language for specifying application ‘policies’, which govern the actions that principals are authorized to perform.
- A language for specifying ‘credentials’, which allow principals to delegate authorization to other principals.
- A ‘compliance checker’, which provides a service to applications for determining how an action requested by principals should be handled, given a policy and a set of credentials.

By design, trust management unifies the notions of security policy, credentials, access control, and authorization. An application that uses a trust-management system can simply ask the compliance checker whether a requested action should be allowed. Furthermore, policies and credentials are written in standard languages that are shared by all trust-managed applications; the security configuration mechanism for one application carries exactly the same syntactic and semantic structure as that of another, even when the semantics of the applications themselves are quite different.

3.4.1 PolicyMaker

PolicyMaker was the first example of a “trust-management engine.” That is, it was the first tool for processing signed requests that embodied the “trust-management” principles articulated in Section

3.4. It addressed the authorization problem directly, rather than handling the problem indirectly via authentication and access control, and it provided an application-independent definition of "proof of compliance" for matching up requests, credentials, and policies. PolicyMaker was introduced in the original trust-management paper by Blaze *et al.* [BFL96], and its compliance-checking algorithm was later fleshed out in [BFS98]. A full description of the system can be found in [BFL96, BFS98], and experience using it in several applications is reported in [BFRS97, LSM97, LMB].

PolicyMaker credentials and policies (collectively referred to as "assertions") are fully programmable: they are represented as pairs (f, s), where s is the *source* of authority, and f is a program describing the nature of the authority being granted as well as the party or parties to whom it is being granted. In a policy assertion, the source is always the keyword **POLICY**. For the PolicyMaker trust-management engine to be able to make a decision about a requested action, the input supplied to it by the calling application must contain one or more policy assertions; these form the "trust root," *i.e.,* the ultimate source of authority for the decision about this request, as shown in Figure 3.3. In a credential assertion, the source of authority is the public key of the issuing entity. Credentials must be signed by their issuers, and these signatures must be verified before the credentials can be used.

PolicyMaker assertions can be written in any programming language that can be "safely" interpreted by a local environment that has to import credentials from diverse (and possibly untrusted) issuing authorities. A version of AWK without file I/O operations and program execution time limits (to avoid denial of service attacks on the policy system) was developed for early experimental work on PolicyMaker (see [BFL96]), because AWK's pattern-matching constructs are a convenient way to express authorizations. For a credential assertion issued by a particular authority to

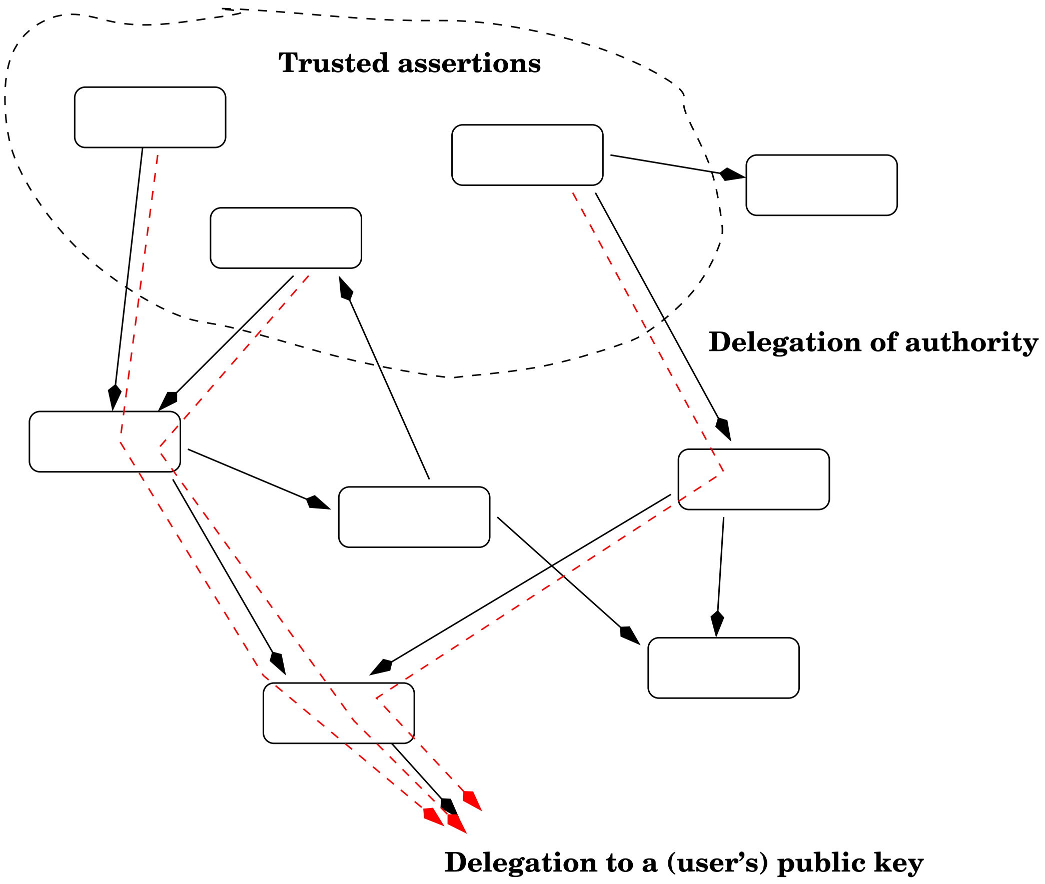


Figure 3.3: Delegation in PolicyMaker, starting from a set of trusted assertions. The dotted lines indicate a delegation path from a trusted assertion (public key) to the user making a request. If all the assertions along that path authorize the request, it will be granted.

be useful in a proof that a request complies with a policy, the recipient of the request must have an interpreter for the language in which the assertion is written (so that the program contained in the assertion can be executed). Thus, it would be desirable for assertion writers ultimately to converge on a small number of assertion languages so that receiving systems have to support only a small number of interpreters and so that carefully crafted credentials can be widely used. However, the question of which languages these will be was left open by the PolicyMaker project. A positive aspect of PolicyMaker's not insisting on a particular assertion language is that all of that work that has gone into designing, analyzing, and implementing the PolicyMaker compliance-checking algorithm will not have to be redone every time an assertion language is changed or a new language is introduced. The "proof of compliance" and "assertion-language design" problems are orthogonal in PolicyMaker and can be worked on independently.

One goal of the PolicyMaker project was to make the trust-management engine as small as possible and analyzable. Architectural boundaries were drawn so that a fair amount of responsibility was placed on the calling application rather than the trust-management engine. In particular, the calling application was made responsible for all cryptographic verification of signatures on credentials and requests. One pleasant consequence of this design decision is that the application developer's choice of signature scheme(s) can be made independently of his choice of whether or not to use PolicyMaker for compliance checking. Another important responsibility that was assigned to the calling application is credential gathering. The input (r, C, P) supplied to the trust-management module is treated as a claim that credential set C contains a proof that request r complies with Policy P. The trust-management module is *not* expected to be able to discover that C is missing just one credential needed to complete the proof and to go fetch that credential from *e.g.,* the corporate database, the issuer's web site, the requester himself, or elsewhere. Later trust-management engines, including KeyNote [BFIK99b, BFIK99c] and REFEREE [CFL$^+$97] divide responsibility between the calling application and the trust-management engine differently than the way PolicyMaker divides it.

The main technical contribution of the PolicyMaker project is a notion of "proof of compliance" that is fully specified and analyzed. We give an overview of PolicyMaker's approach to compliance checking here; a complete treatment of the compliance checker can be found in [BFS98].

The PolicyMaker runtime system provides an environment in which the policy and credential assertions fed to it by the calling application can cooperate to produce a proof that the request complies with the policy (or can fail to produce such a proof). Among the requirements for this cooperation are a method of inter-assertion communication and a method for determining that assertions have collectively succeeded or failed to produce a proof.

Inter-assertion communication in PolicyMaker is done via a simple, append-only data structure on which all participating assertions record intermediate results. Specifically, PolicyMaker initializes the proof process by creating a "blackboard" containing only the request string r and the fact that no assertions have thus far approved the request or anything else. Then PolicyMaker runs the various assertions, possibly multiple times each. When assertion (f_i, s_i) is run, it reads the contents of the blackboard and then adds to the blackboard one or more *acceptance records* (i, s_i, R_{ij}). Here R_{ij} is an application-specific action that source s_i approves, based on the partial proof that has been constructed thus far. R_{ij} may be the input request r, or it may be some related action that this application uses for inter-assertion communication. Note that the meanings of the action strings R_{ij} are understood by the application-specific assertion programs f_i, but they are not understood by PolicyMaker. All PolicyMaker does is run the assertions and maintain the global blackboard, making sure that the assertions do not erase acceptance records previously written by other assertions, fill up the entire blackboard so that no other assertions can write, or exhibit any other non-cooperative behavior. PolicyMaker never tries to interpret the action strings R_{ij}.

A proof of compliance is achieved if, after PolicyMaker has finished running assertions, the blackboard contains an acceptance record indicating that a policy assertion approves the request r. Among the nontrivial decisions that PolicyMaker must make are (1) in what order assertions should be run, (2) how many times each assertion should be run, and (3) when an assertion should be discarded because it is behaving in a non-cooperative fashion. Blaze *et al.* [BFS98] provide:

- A mathematically precise formulation of the PolicyMaker compliance-checking problem.
- Proof that the problem is undecidable in general and is NP-hard even in certain natural special cases.
- One special case of the problem that is solvable in polynomial-time, is useful in a wide variety of applications, and is implemented in the current version of PolicyMaker.

Although the most general version of the compliance-checking problem allows assertions to be arbitrary functions, the computationally tractable version that is analyzed in [BFS98] and implemented in PolicyMaker is guaranteed to be correct only when all assertions are monotonic. (Basically, if a monotonic assertion approves action a when given evidence set E, then it will also approve action a when given an evidence set that contains E; see [BFS98] for a formal definition.) In particular, correctness is guaranteed only for monotonic *policy* assertions, and this excludes certain types of policies that are used in practice, most notably those that make explicit use of "negative credentials" such as revocation lists. Although it is a limitation, the monotonicity requirement has certain advantages. One of them is that, although the compliance checker may not handle all potentially desirable policies, it is at least analyzable and provably correct on a well-defined class of policies. Furthermore, the requirements of many non-monotonic policies can often be achieved by monotonic policies. For example, the effect of requiring that an entity *not* occur on a revocation list can also be achieved by requiring that it present a “certificate of non-revocation”; the choice between these two approaches involves trade-offs among the (system-wide) costs of the two kinds of credentials and the benefits of a standard compliance checker with provable properties. Finally, restriction to monotonic assertions encourages a conservative, prudent approach to security: In order to perform a potentially dangerous action, a user must present an adequate set of affirmative credentials; no potentially dangerous action is allowed “by default,” simply because of the absence of negative credentials.

3.5 The KeyNote Trust-Management System

KeyNote [BFIK99b, BFIK99c] was designed according to the same principles as PolicyMaker, using credentials that directly authorize actions instead of dividing the authorization task into authentication and access control. Two additional design goals for KeyNote were standardization and ease of integration into applications. To address these goals, KeyNote assigns more responsibility to the trust-management engine than PolicyMaker does and less to the calling application; for example, cryptographic signature verification is done by the trust-management engine in KeyNote and by the application in PolicyMaker. KeyNote also requires that credentials and policies be written in a specific assertion language, designed to work smoothly with KeyNote’s compliance checker. By fixing a specific assertion language that is flexible enough to handle the security policy needs of different applications, KeyNote goes further than PolicyMaker toward facilitating efficiency, interoperability, and widespread use of carefully written credentials and policies, at the cost of reduced

```
KeyNote-Version: 2
Authorizer: "rsa-hex:1023abcd"
Licensees: "dsa-hex:986512a1" || "rsa-hex:19abcd02"
Comment: Authorizer delegates read access to either of the
         Licensees
Conditions: (file == "/etc/passwd" &&
             access == "read") -> "true";
Signature: sig-rsa-md5-hex:"f00f5673"
```

Figure 3.4: Sample KeyNote assertion authorizing either of the two keys appearing in the Licensees field to read the file "/etc/passwd".

expressibility and interaction between different policies (compared to PolicyMaker).

A calling application passes to a KeyNote evaluator a list of credentials, policies, and requester public keys, and an "Action Attribute Set." This last element consists of a list of attribute/value pairs, similar in some ways to the $Unix^{TM}$ shell environment (described in the *environ(7)* manual page of most Unix installations). The action attribute set is constructed by the calling application and contains all information deemed relevant to the request and necessary for the trust decision. The action-environment attributes and the assignment of their values must reflect the security requirements of the application accurately. Identifying the attributes to be included in the action attribute set is perhaps the most important task in integrating KeyNote into new applications. The result of the evaluation is an application-defined string (perhaps with some additional information) that is passed back to the application. In the simplest case, the result is of the form "authorized."

The KeyNote assertion format resembles that of e-mail headers. An example (with artificially short keys and signatures for readability) is given in Figure 3.5.

As in PolicyMaker, policies and credentials (collectively called assertions) have the same format. The only difference between policies and credentials is that a policy (that is, an assertion with the keyword **POLICY** in the *Authorizer* field) is locally trusted (by the compliance-checker) and thus needs no signature.

KeyNote assertions are structured so that the *Licensees* field specifies explicitly the principal or principals to which authority is delegated. Syntactically, the Licensees field is a formula in which the arguments are public keys and the operations are conjunction, disjunction, and threshold. The semantics of these expressions are specified in Appendix A.

The programs in KeyNote are encoded in the *Conditions* field and are essentially tests of the action attributes. These tests are string comparisons, numerical operations and comparisons, and pattern-matching operations.

We chose such a simple language for KeyNote assertions for the following reasons:

- AWK, one of the first assertion languages used by PolicyMaker, was criticized as too heavyweight for most relevant applications. Because of AWK's complexity, the footprint of the interpreter is considerable, and this discourages application developers from integrating it into a trust-management component. The KeyNote assertion language is simple and has a small-size interpreter.

- In languages that permit loops and recursion (including AWK), it is difficult to enforce resource-usage restrictions, but applications that run trust-management assertions written by unknown sources often need to limit their memory and CPU usage.

  We believe that for our purposes a language without loops, dynamic memory allocation, and certain other features is sufficiently powerful and expressive[BIK01b, BIK01a]. The KeyNote assertion syntax is restricted so that resource usage is proportional to the program size. Similar concepts have been successfully used in other contexts [HKM$^+$98].

- Assertions should be both understandable by human readers and easy for a tool to generate from a high-level specification. Moreover, they should be easy to analyze automatically, so that automatic verification and consistency checks can done. This is currently an area of active research.

- One of our goals is to use KeyNote as a means of exchanging policy and distributing access control information otherwise expressed in an application-native format. Thus the language should be easy to map to a number of such formats (*e.g.,* from a KeyNote assertion to packet-filtering rules).

- The language chosen was adequate for KeyNote's evaluation model.

This last point requires explanation.

In PolicyMaker, compliance proofs are constructed via repeated evaluation of assertions, along with an arbitrated "blackboard" for storage of intermediate results and inter-assertion communication.

In contrast, KeyNote uses an algorithm that attempts (recursively) to satisfy at least one policy assertion. Referring again to Figure 3.3, KeyNote treats keys as vertices in the graph, with (directed) edges representing assertions delegating authority. In the prototype implementation, we used a Depth First Search algorithm, starting from the set of trusted ("POLICY") assertions and trying to construct a path to the key of the user making the request. An edge between two vertices in the graph exists only if:

- There exists an assertion where the *Authorizer* and the *Licensees* are the keys corresponding to the two vertices.

- The predicate encoded in the *Conditions* field of that KeyNote assertion authorizes the request.

Thus, satisfying an assertion entails satisfying both the *Conditions* field and the *Licensees* key expression. Note that there is no explicit inter-assertion communication as in PolicyMaker; the *acceptance records* returned by program evaluation are used internally by the KeyNote evaluator and are never seen directly by other assertions. Because KeyNote's evaluation model is a subset of PolicyMaker's[5], the latter's compliance-checking guarantees are applicable to KeyNote. Whether the more restrictive nature of KeyNote allows for stronger guarantees to be made is an open question requiring further research.

Ultimately, for a request to be approved, an assertion graph must be constructed between one or more policy assertions and one or more keys that signed the request. Because of the evaluation model, an assertion located somewhere in a delegation graph can effectively only refine (or pass on) the authorizations conferred on it by the previous assertions in the graph. (This principle also holds for PolicyMaker.) For more details on the evaluation model, see Appendix A.

It should be noted that PolicyMaker's restrictions regarding "negative credentials" also apply to KeyNote. Certificate revocation lists (CRLs) are not built into the KeyNote (or the PolicyMaker) system; these can be provided at a higher (or lower) level, perhaps even transparently to KeyNote[6]. The problem of credential discovery is also not explicitly addressed in KeyNote.

Finally, note that KeyNote, like other trust-management engines, does not directly *enforce* policy; it only provides "advice" to the applications that call it. KeyNote assumes that the application itself is trusted and that the policy assertions are correct. Nothing prevents an application from submitting misleading assertions to KeyNote or from ignoring KeyNote altogether.

[5]In PolicyMaker, the programs contained in the assertions and credentials can interact with each other by examining the values written on the blackboard, and reacting accordingly. This, for example, allows for a negotiation of sorts: "I will approve, if you approve", "I will also approve if you approve", "I approve", "I approve as well"...In KeyNote, each assertion is evaluated exactly once, and cannot directly examine the result of another assertion's evaluation.

[6]Note that the decision to consult a CRL is (or should be) a matter of local policy.

Chapter 4

STRONGMAN Architecture

The concerns outlined in Section 1.2 guide the design of our access control architecture. In our system, users will be identified by their public keys (each user may have multiple keys, for different purposes/applications). These public keys are used in the context of various protocols to authenticate the users to specific services.

Our system must effectively scale in two different, but related, areas: system and management complexity (and size).

Addressing system complexity requires policy specification, distribution, and enforcement *mechanisms* that can handle large numbers of users, enforcement points, and applications. Furthermore, the system must be able to handle the increased complexity of mechanism interactions. As we saw in Section 2, none of the systems that have been proposed so far can effectively address this issue. In particular, fully-centralized approaches (Figure 4.2) demonstrate poor scaling properties; systems where policy is centrally specified but distributed (synchronously) to all enforcement points (Figure 4.3) require these to maintain large amounts of potentially unneeded state, and require communication as a result of frequent security operations such as adding/removing users or modifying their privileges. Fully-decentralized solutions (Figure 4.4) do not easily allow for interaction between different applications, unless a “coordinating” layer is added on top of the existing system (effectively turning the system into a centralized one)[1].

As a first design choice then, our system must exhibit the scaling properties of a decentralized policy specification, distribution, and enforcement system, while retaining the ability to let different applications and protocols interact as needed. Therefore, policy should be expressed in a way that is easy to distribute to enforcement points “on the fly”, and which is easy for the enforcement points to verify and process efficiently. One way of expressing low-level policy is in the form of public-key credentials, as we saw in Chapter 3: an administrator can issue signed statements that contain the privileges of users; enforcement points can verify the validity of these credentials and enforce the policies encoded therein.

This approach offers some additional benefits: since credentials are integrity-protected via a digital signature, they need not be protected when transmitted over the network (thus avoiding a potential security bootstrap problem). Thus, it is possible to distribute policies in any of the following three ways:

1. Have the policies “pushed” directly to the enforcement points. While this is the simplest approach, it requires that all the policy information an enforcement point is going to need to know be stored locally. As a simple exercise, assume a corporate web server that any of 100,000 users may access; identifying each user would require knowledge of their public key, for authentication purposes. Assuming a typical RSA key of 128 bytes (1024 bits), simply storing this information on the web server would require approximately 13 MB of

[1]We should note that no real system follows any of these three approaches (especially the centralized ones) in their pure form. For example, centralized systems typically allow for policy decisions to be cached at the enforcement points, or they employ a hierarchy of policy decision points, for improved performance.

storage, excluding any access control information. Typical certificate encodings typically triple to quadruple the size of the per-user information that is needed, and the actual access control information will further add to this. While a web server (and most general-purpose computer systems these days) can afford such storage requirements, embedded systems or routers do not have any persistent storage capacity of that kind.

Furthermore, under this scheme, changes in the policy (*e.g.,* adding a new user) require all affected systems to be contacted and their local copy of the policy updated. If such changes are frequent, or the number of affected systems is large, the cost can prove prohibitive.

Finally, the enforcement point will also have to incur a processing cost for examining potentially "useless" policy entries when trying to determine whether a specific user request should be granted. The exact cost depends on the particular scheme used to store and process this information.

2. Have the policies "pulled" by the enforcement points from a policy repository as needed, and then stored locally. This exhibits much better behavior in terms of processing and storage requirements, but requires that the enforcement point perform some additional processing (and incur some communication overhead) when evaluating a security request. System availability can be addressed via replicated repositories; an attacker that compromises one or more of these can deny service to legitimate users, but cannot otherwise affect a policy decision.

   This approach offers two additional advantages: first, it is relatively easy to deploy since it requires modification of only the enforcement points (as opposed to modifying all the clients and other network elements). Secondly, it effectively addresses privilege revocation (which we discuss later in this section).

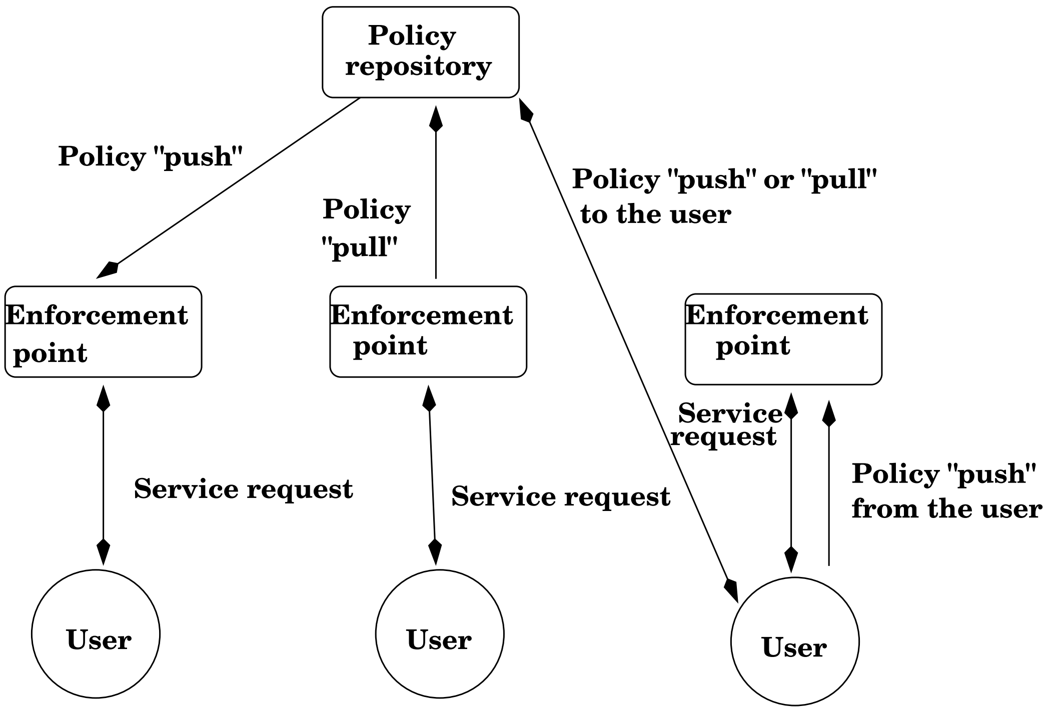


Figure 4.1: Policy distribution: policy is pushed to the enforcement points, policy is "pulled" by the enforcement points from a repository, and policy is supplied to the end users which have to deliver it to the enforcement points as needed.

3. Have the policies distributed to the client (user) systems, and make these responsible for delivering them to the enforcement points. While this approach requires modification of the client, most security protocols already provide certificate exchange as part of the authentication mechanism; as we shall see in Chapter 5, it is relatively straightforward to modify such protocols to deliver the kind of credentials used in our system instead. Furthermore, since the end systems hold all the credentials that are relevant to them, it is possible to determine in advance under what conditions a request will be granted by an enforcement point (*e.g.,* how strong the encryption should be to be able to see confidential information on the corporate web server).

The three approaches to policy distribution are shown in Figure 4.1. In practice, a combination of (2) and (3) will be used in our system: if the client system provides credentials during the authentication phase, these are used to determine the user's privileges; otherwise, the system may contact a repository to retrieve the relevant information or, if it is overloaded, deny the request and ask that the user provide the missing information in a subsequent request. One advantage of this approach is that policy can be treated as "soft state," and periodically be purged to handle new users and requests (using LRU, or some other replacement mechanism). If the policy is needed again, it will be re-instantiated. This mechanism is very similar to the memory page-fault mechanism used by modern operating systems, and thus all the studies on page replacement techniques are applicable here. Because of these similarities, we call our mechanism "lazy policy instantiation."

One benefit of our choice of credentials as a means for distributing policy is the fact that one of the frequently-done operations (adding a user, or giving additional privileges to an existing user) is cheap: we simply have to issue the necessary credentials for the user in question, and make them available in the repository. Under any of the distribution schemes already described, the new policy will take effect as soon as the next request that requires is appears.

On the other hand, one other frequent operation (removing a user, or revoking some an existing user's privileges) is more complicated in an environment where policy is not centrally stored and maintained. We defer discussion of this issue until Section 4.2.

The second scale-related problem area our system must address is administrative complexity; the increased system scale stretches the ability of human administrators to handle its complexity. One well-known and widely used solution is that of "separation of duty": different administrators are made responsible for managing different aspects of the larger system. In computer networks, this separation can be implemented across network boundaries (*e.g.,* LAN or WAN administrators)

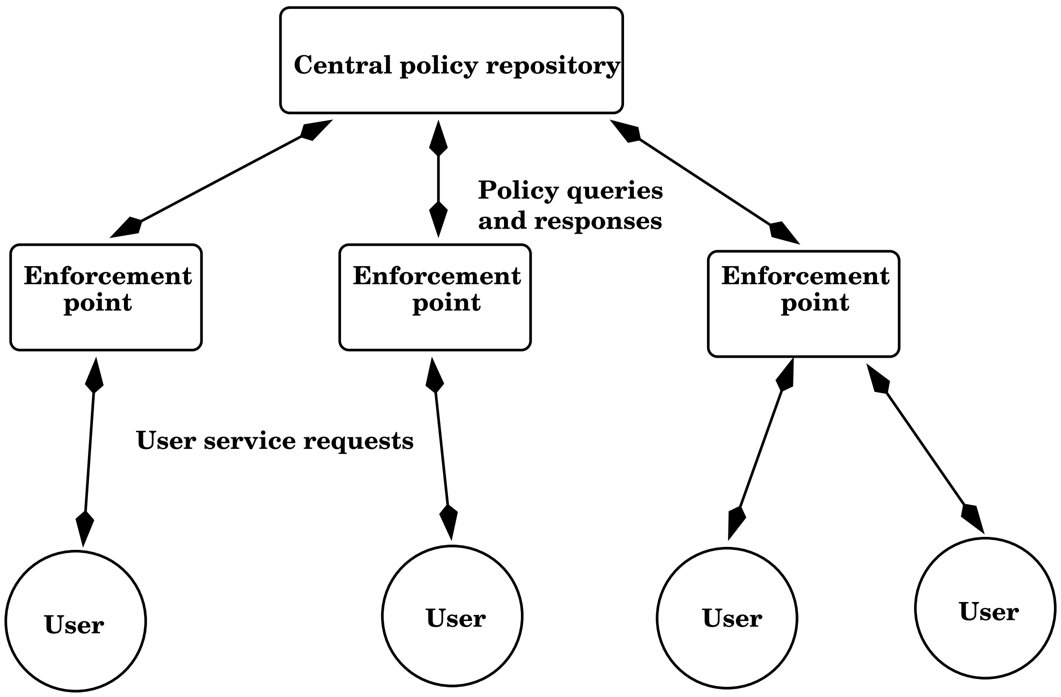


Figure 4.2: Centralized policy specification and enforcement: the enforcement points contact the server with the user request details, and expect an answer. Policy evaluation is done at the central repository, for each request. Responses may be cached at the enforcement points, as long as the details of the request do not change; systems implementing this approach must also address policy consistency issues. Interactions between services and protocols are easy to define, since all the information is centrally available.

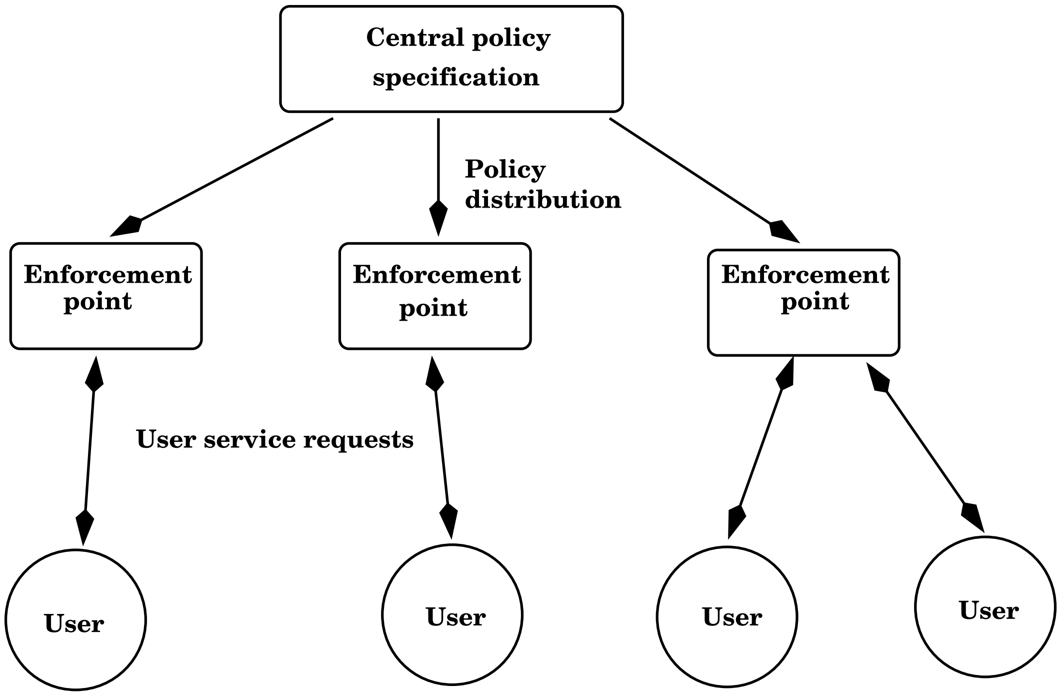


Figure 4.3: Central policy specification, decentralized enforcement: policy for all users, enforcement points, and applications is specified simultaneously. The results are then distributed at all the (relevant) enforcement points. Changes to the running system require communication with the affected enforcement points. Interactions between protocols and services are easy to define, since all the information is centrally available.

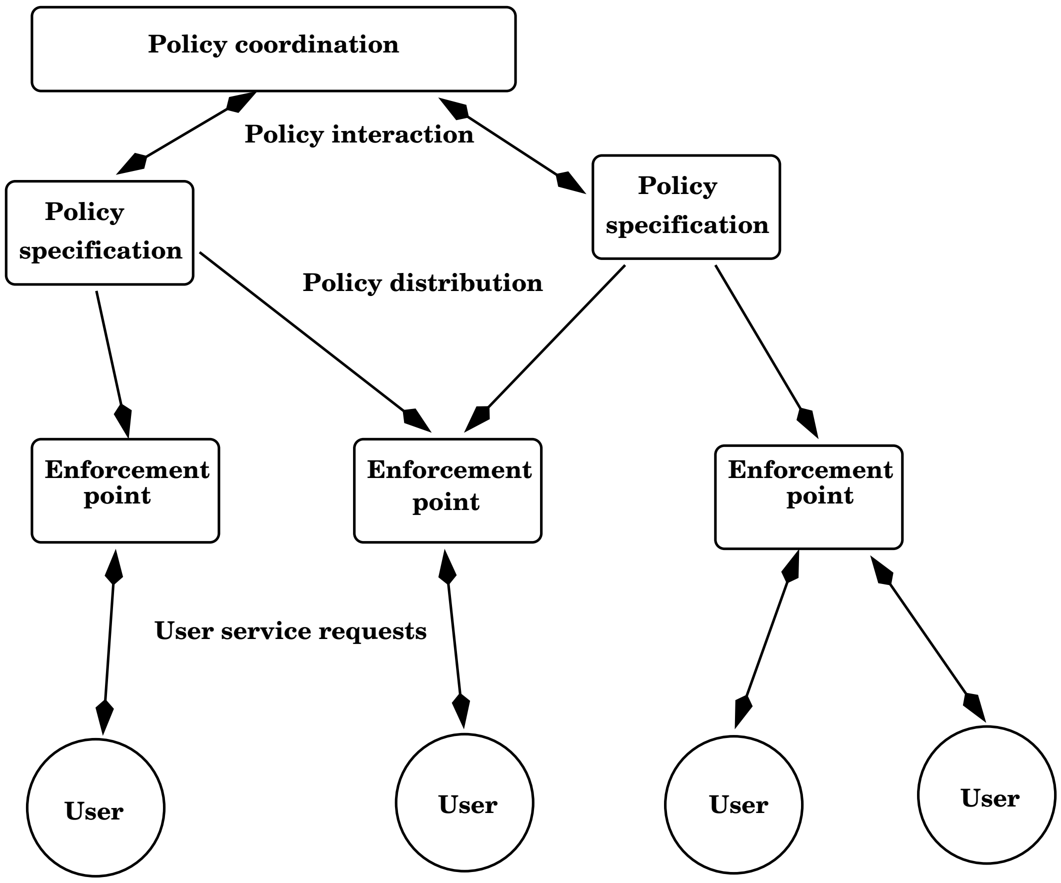


Figure 4.4: Decentralized policy specification and enforcement: policy is specified by different administrators for the different applications, users, and enforcement points. Policy may be distributed directly to the enforcement points, or may be made available to the users in the form of certificates or tickets. Interactions between protocols and services are difficult to express, unless an additional "coordination" layer is added, which re-introduces a measure of centralization to the system; the coordination layer maybe explicit (in the form of a meta-policy server), or implicit (in the form of a meta-policy language).

or across application boundaries (*e.g.,* different administrators for the firewalls, the web servers, the print servers, *etc.*). Multiple layers of management may be used, to handle increasing scale. Thus, our system must support this management approach. One commonly-used mechanism that implements hierarchical management in decentralized systems is delegation of authority.

Note that the degree of (de)centralization of policy specification and enforcement are independent of each other: decentralized policy specification may be built on top of a centralized enforcement system, by providing a suitable interface to the different administrators; similarly, a centralized policy specification system can easily be built on top of decentralized enforcement architecture, as shown in Figure 4.5.

These considerations argue for a multi-layer design, such as shown in Figure 4.8. Administrators can use any number of different interfaces in specifying access control policy. Thus, administrators can pick an interface they are already familiar with or one that is not very different from what they have been using. Furthermore, it is possible to construct application-specific interfaces, that capture the particular nuances of the application they control. We discuss how different administrators, using different management front-ends, can effectively coordinate and express dependencies across different protocols in Section 4.1.

This architecture has an intentional resemblance to the IP "hourglass", and resolves heterogeneity in similar ways, *e.g.,* the mapping of the interoperability layer onto a particular enforcement device, or the servicing of multiple applications with a policy *lingua franca*.

Is is important to realize that the design in Figure 4.8 refers to the logical flow of policy; the system itself follows the decentralized policy specification and enforcement approach. High-level policy is specified separately by each administrator; we shall see examples of such policies in Chapter 5. This interface takes as input the stated policy and information from a network/user

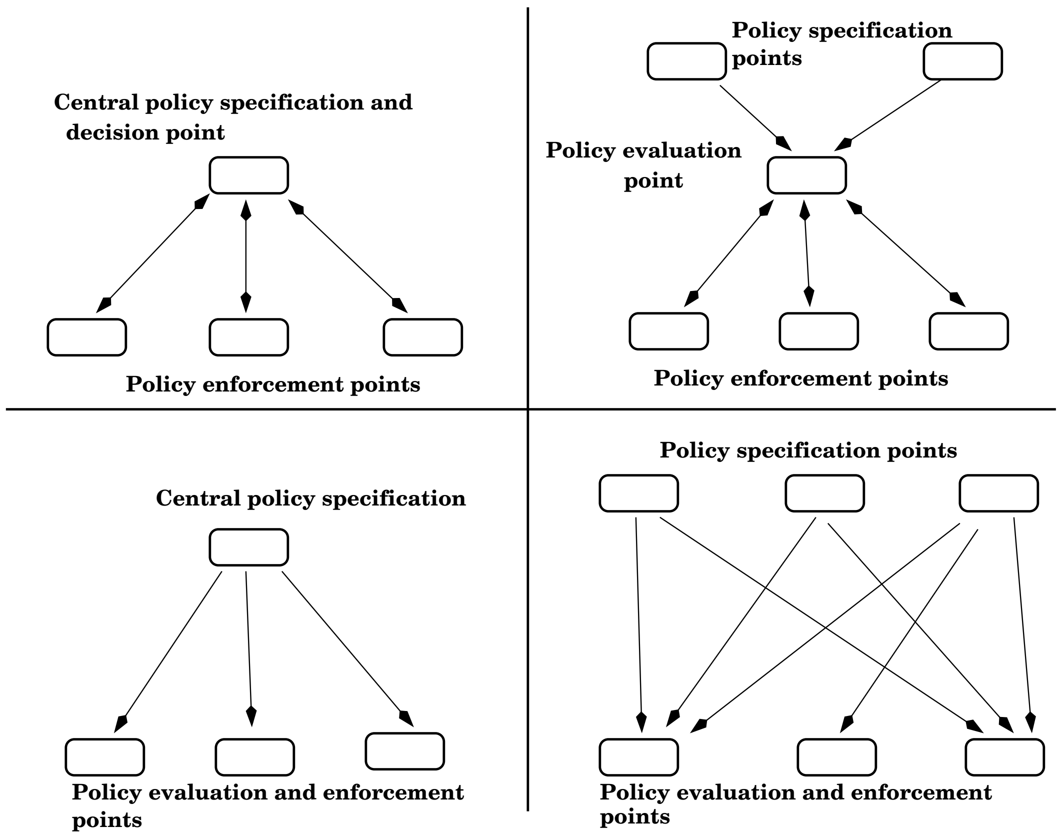

Central policy specification and decision point
Policy enforcement points
Policy specification points
Policy evaluation point
Policy enforcement points
Central policy specification
Policy evaluation and enforcement points
Policy specification points
Policy evaluation and enforcement points

Figure 4.5: The different combinations of policy specification and decision making with respect to (de)centralization. Although the actual enforcement is done at the different network elements (denoted as "enforcement points"), enforcement typically refers to the decision making (policy evaluation).

database, and produces policy statements in the common language of the low-level policy system. Thus, the low-level policy system (the policy interoperability layer, as it were) must be powerful and flexible enough to handle different applications. These low-level policy statements are then distributed *on-demand* to the enforcement points, where policy evaluation and enforcement is performed locally.

In order to accommodate management delegation, one of two approaches may be taken: delegation may be implemented as part of the low-level policy mechanism, or as a function of the high-level policy specification system, as shown in Figures 4.7 and 4.6. The high-level approach offers considerable flexibility in expressing delegation and related restrictions, but causes the higher echelons of the administrative hierarchy to become bottlenecks, since they have to be involved in all policy specification. Thus, we have opted for a design where delegation is handled by the low-level policy system. One advantage of this is that administration hierarchies can be built "on the fly", simply by delegating to a new administrator.

To summarize, our choice for a low-level policy mechanism is dictated by:

1. Flexibility in the types of applications it can support.

2. Efficiency in evaluating policy.

3. Ability to naturally and efficiently express and handle delegation of authority.

4. Simplicity, as a desirable property of any system[2].

The KeyNote trust-management system we discussed in Section 3.5 exhibits these properties, and is used as the low-level policy mechanism in STRONGMAN. In the following two sections,

[2]To paraphrase Albert Einstein, "every system should be as simple as possible, but no more."

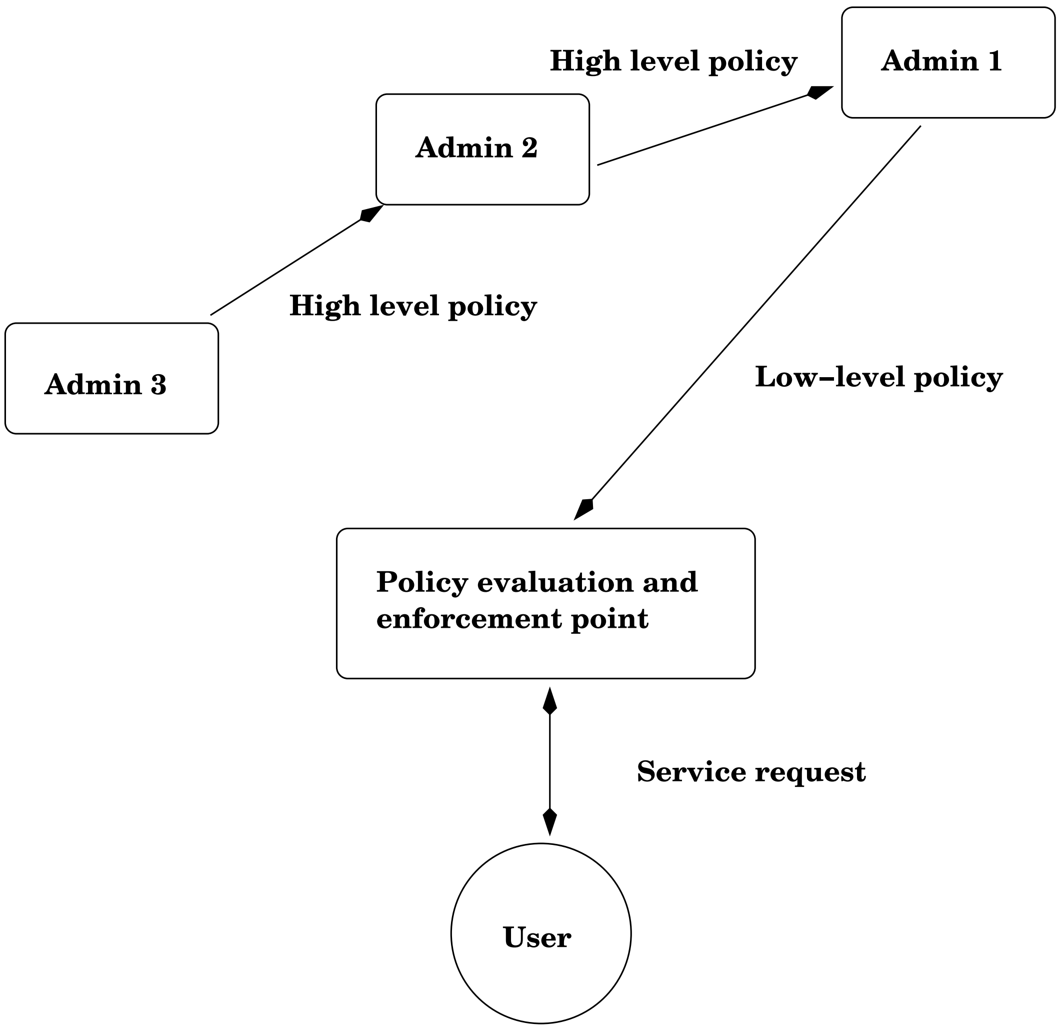


Figure 4.6: Delegation as a function of high-level policy specification. High-level policy statements by different administrators at level N of the management hierarchy are imported and combined at level *N-1*, recursively. The top-level administrator produces the final low-level policy statement, as a result of the composition of all the policies.

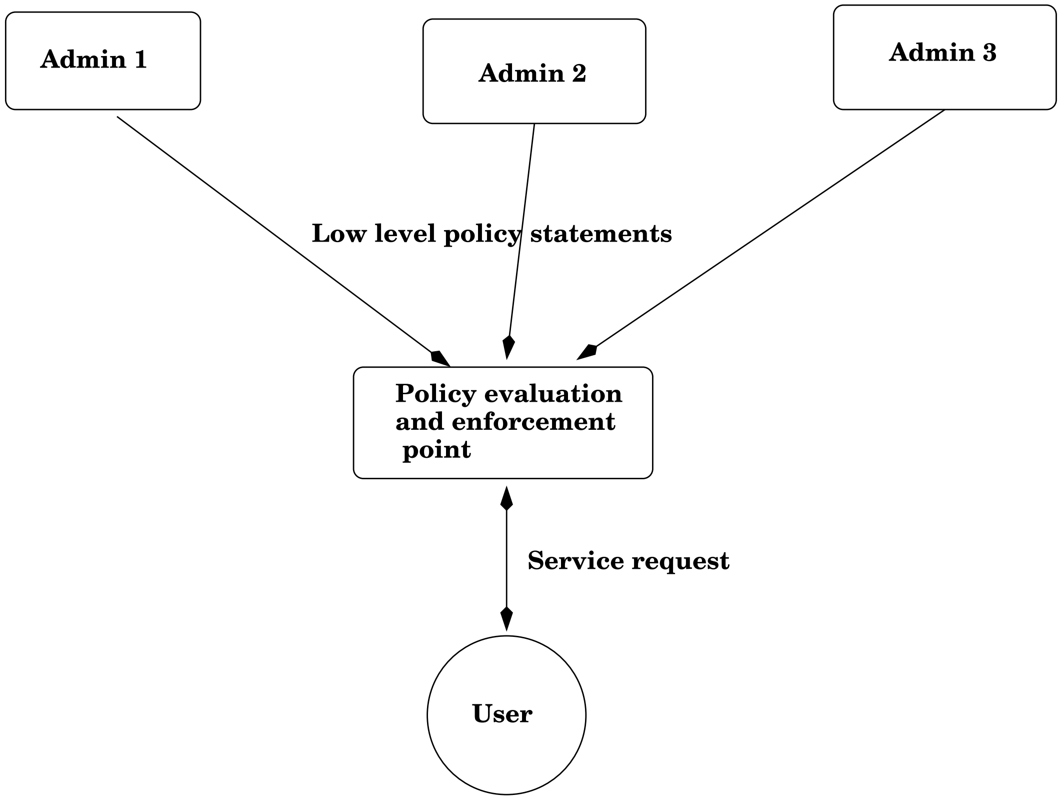

Admin 1
Admin 2
Admin 3
Low level policy statements
Policy evaluation and enforcement point
Service request
User

Figure 4.7: Delegation as part of the low-level policy system. Low-level policy statements from all (relevant) administrators are combined at the policy evaluation point.

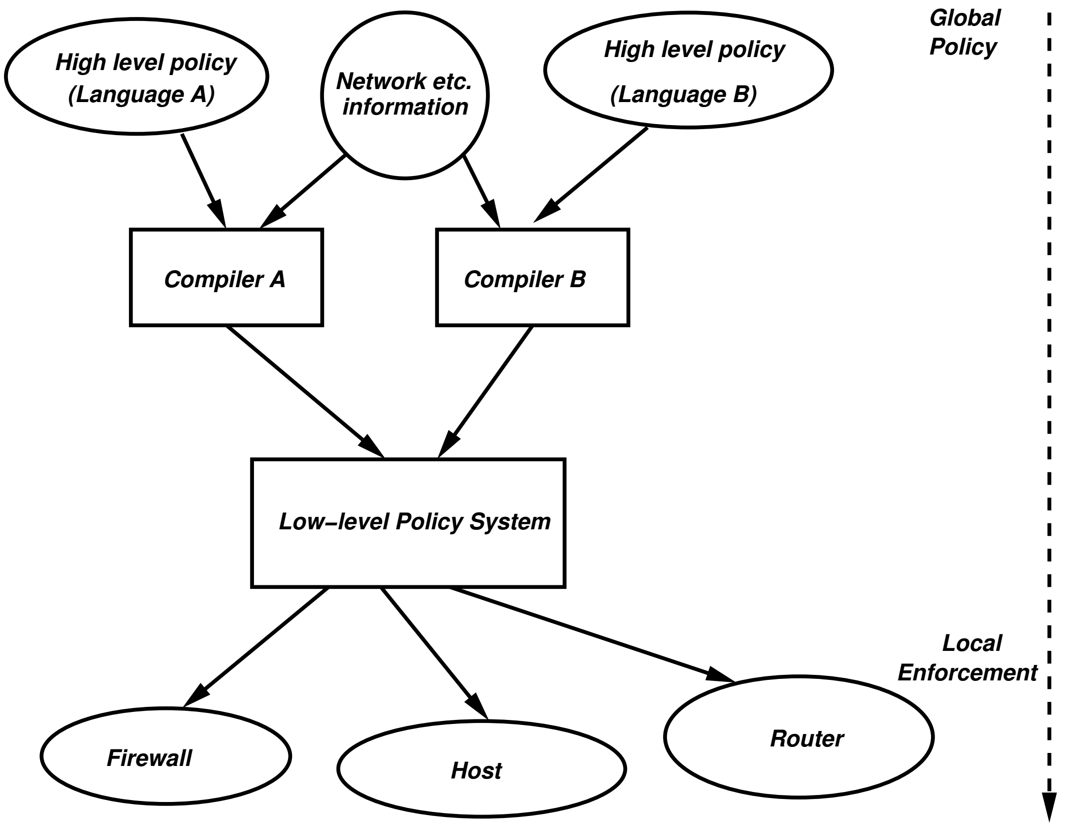


Figure 4.8: The STRONGMAN architecture.

we examine policy composition (how policy statements issued by different administrators are combined) and privilege revocation in STRONGMAN.

4.1 Distributed Management and Policy Composition

As we already described, each administrator issues policy statements (using some high-level configuration mechanism) that have to be combined at the policy evaluation points, to implement a distributed, hierarchical management structure.

As the reader will recall from our discussion of KeyNote in Section 3.5, arbitrary graphs of authority delegation may be built using KeyNote, as shown in Figure 4.9. Each administrator is represented by a public key, and each edge in the tree represents a credential from one administrator to another, delegating authority. At the leaves of the tree, administrators authorize users to access particular services under specific (application-dependent) conditions (*e.g.,* a URL is served only if the connection is encrypted).

Policy enforcement points trust the root of the tree to define policy. The root authority may be split among different administrators: one way of configuring the system is to have multiple top-level administrators each of which can issue policy statements independently of the others; another way is to require a majority (or other threshold) of administrators to delegate authority before delegation takes effect. Both of these conditions can be expressed easily in KeyNote:

```
Authorizer: ``POLICY''
Licensees: Top_Admin_1_Public_Key || Top_Admin_2_Public_Key ||
           Top_Admin_3_Public_Key


Authorizer: ``POLICY''
Licensees: 2-of(Top_Admin_1_Public_Key, Top_Admin_2_Public_Key,
                Top_Admin_3_Public_Key)
```

At each level, administrators can delegate authority to other administrators or users, while retaining the ability to restrict the scope of the delegation. For a request to be granted, all the policy statements (KeyNote credentials) that link the user requesting access with the top-level administrator(s) have to be present, and all these statements have to authorize the request. Rogue or subverted administrators or users can only affect the part of the hierarchy for which they are the root; invalid or improperly scoped statements by an administrator (*e.g.,* policy statements about firewall access control from the web administrator) are ineffective, since said administrator is not

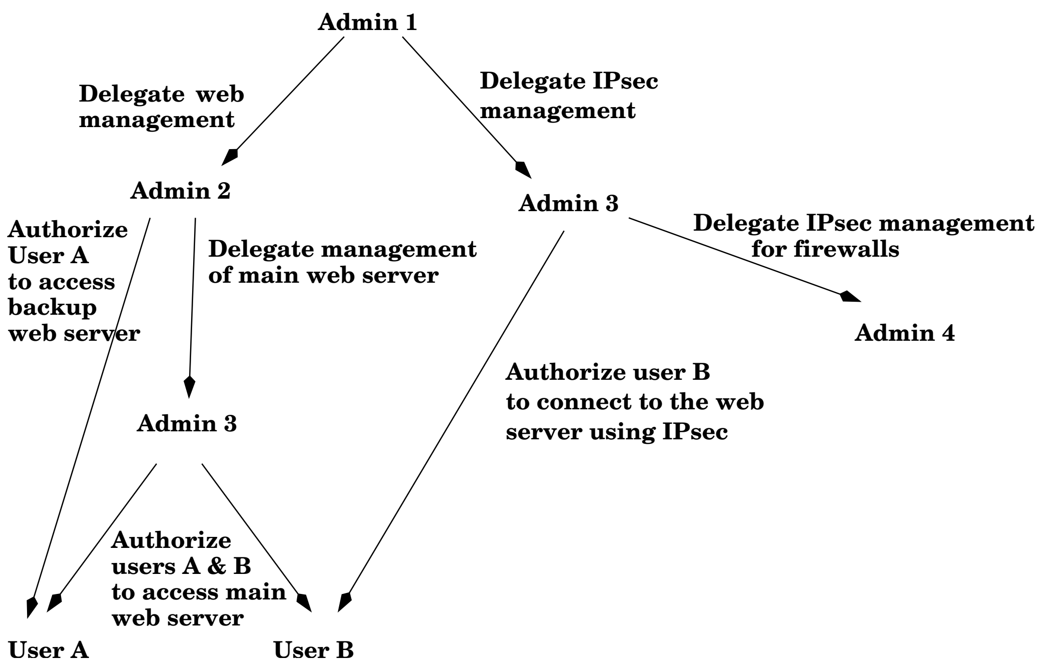

Admin 1
Delegate web management
Delegate IPsec management
Admin 2
Admin 3
Delegate IPsec management for firewalls
Admin 4
Authorize User A to access backup web server
Delegate management of main web server
Admin 3
Authorize user B to connect to the web server using IPsec
Authorize users A & B to access main web server
User A
User B

Figure 4.9: Delegation using KeyNote.

authorized to issue such statements. Enforcement of this property is done by KeyNote, as part of the policy evaluation algorithm we discussed in Section 3.5. Thus, in STRONGMAN, separation of duty is enforced by the hierarchy.

The scope of the separation is encoded in the KeyNote credentials that are produced from the high-level policy statements issued by the administrators: administrators may be authorized to issue policy statements for different enforcement points, specifying these enforcement points in the credentials issued to these administrators; alternatively, different administrators may be authorized to issue policy statements for different services. Both of these scenarios are shown in Figure 4.9. It is possible to combine the two approaches in the same management hierarchy, to address management bottlenecks.

At each layer, an administrator simply needs to know the administrators delegating authority to her, as well as the administrators or users she is delegating authority to. In this way, arbitrarily complicated authorization graphs may be constructed, mapping any chain of command. In scenarios where controlled exposure of resources is required between different administration domains (*e.g.,* an industrial partnership), administrators of each domain can simply treat the respective administrators of the other domain as users of their system, and delegate to them privileges that can then be re-delegated to the users of that domain.

Finally, recall that one of the requirements for our system is that it must be possible for the administrators to express dependencies across applications. For example, access to a specific URL may depend on the security level of the underlying network connection; in our example, the network connection can be protected by SSL at the application layer, or by IPsec at the network layer. The high-level policy specification should be capable of expressing such dependencies irrespective of the exact form used to express these (high-level) policies. This imposes the requirement that our approach be simple enough such that it can be integrated in any of the different configuration mechanisms, be they language or GUI-based. Furthermore, similar to our discussion about delegation implemented as a function of the high-level or the low-level policy system, our system should implement this interaction at the policy interoperability layer.

To this end, we introduce the concept of "composition hook"; the administrator of one application can issue policy statements that depend on the result of the policy evaluation of another application (in our example above, the two applications would be web access and IPsec access control). These composition hooks are translated to low-level policy conditions (as part of the KeyNote Conditions field), and are operated on in the same way as other access control variables. We shall see examples of this in Chapter 5.

For this scheme to work, the "other" application (in our example, IPsec) must generate this information (the composition hook), and then have this information propagate across different enforcement layers. In the case of the network stack, this can be done by extending the various APIs that are used between the different network layers. To implement our example, we extended the information available through the unix *getsockopt ()* system call (which gives the caller information about a network connection) to also return the composition hook from the IPsec layer. Thus, the composition information becomes available to the policies as these policies are evaluated, without any need for coordinating policies between administrators. One drawback of this approach is that the APIs have to be extended to accommodate for this information flow. Fortunately, new protocols and services are not introduced very often, and extending the relevant APIs can be done at the time of deployment. As we shall see in Chapter 5, the only coordination necessary between administrators is an agreement (or understanding) on the meaning of the values of the composition hooks emitted by the various policies.

The same approach can be used to propagate information between different network elements, *e.g.,* a firewall and an end host: the end host could ask the firewall whether a particular connection was cryptographically protected as it arrived at the firewall. The communication between the firewall and the end host must be secured, which adds some complexity to the system, also increasing the work load of the firewall. Finally, there is a widely-held belief that security ought to be an end-to-end property, thus reducing the need for such a "security properties" signaling mechanism. For these reasons, we decided not to further investigate this extension.

4.2 Policy Updates and Revocation

Adding a new user or granting more privileges to an existing user is simply a matter of issuing a new credential (note that both operations are equivalent in terms of sequence of operations in our system).

The inverse operation, removing a user or revoking issued privilege, means notifying entities that might try to use it that the information in it is no longer valid, even though the credential itself has not expired. Potential reasons for the revocation include theft or loss of the administrator key used to sign the credential (in which case, all certificates signed by that key need to be revoked), theft or loss of the user or administrator key authority has been delegated to, or discovery that the information contained in the certificate has become inaccurate.

There are four main mechanisms for certificate revocation:

1. The validity period of the credential itself; if it is set to a sufficiently small value, then the window of revocation is effectively limited to that. On the other hand, a short lifetime means that the a user's credential has to be re-issued much more often, which implies increased work for the administrator (in terms of credential generation and distribution). In the extreme case, where credentials are made valid for a few minutes only, the CA is effectively involved in (almost) every authentication protocol exchange.

   This approach works well when credentials are used in a transient manner (*e.g.,* to authorize temporary access to a resource). On the other hand, if credential revocation is rare in a given deployed system, the amount of unnecessary work done by the system (re-issuing short lived policy statements) can be quite high.

2. Certificate revocation lists (CRLs), and their variants. The idea is that the administrator compiles a list of credentials that must be revoked, and distributes this to the enforcement points (or, as is more typical, the enforcement points periodically retrieve the list from a repository). The CRL is signed by the administrator, and contains a timestamp. An enforcement point

can verify that it has received a valid and reasonably recent copy of the CRL by verifying the signature and examining the timestamp. Revoked credentials can be removed from the CRL as soon as their validity period expires.

This approach works well when, on the average, only a small number of credentials are revoked. Various approaches, such as Delta-CRLs or Windowed Revocation[MJ00], attempt to address this scalability issue.

Note that, while it is the credential verifier that needs to examine a copy of the CRL, the authentication protocol can require the client system to obtain and supply a reasonably recent CRL to the responder. This may be particularly useful if the responder is an overloaded web server *etc.*

3. Refresher credentials. In this scheme, the owner of a long-lived credential has to periodically retrieve a short-lived credential that must be used in conjunction with the long-lived one. They can do this by simply contacting the issuer of the credential (or some other entity that handles refresher credentials). The advantage of this approach over direct short-lived credentials is that a refresher credential is only issued if a user actually needs one. On the other hand, it requires some communication on the part of the credential owner (as do all revocation schemes, except lifetime-based revocation).

4. Online certificate-status protocols, such as OCSP[MAM$^+$99], have the credential verifier query the credential issuer (or other trusted entity) about the validity of a credential. One drawback is that it is the verifier that must do this status check; if the verifier is a web server (as would be typically the case), this approach places additional burden on it. On the other

hand, this approach does not require even roughly synchronized clocks, as do solutions (1)—(3). However, since the exchange needs to be secured, the protocol can be fairly expensive.

In cases (2)—(4), the credential issuer (or other trusted entity) must issue statements as to the validity of an issued credential. Since these statements must be verifiable, these approaches require that this issuer's private key is available online (especially for cases (3) and (4)). However, we can use separate keys for issuing and revoking credentials; both keys can be present in the credential. In the event that the machine where the revoking key is stored is compromised, an attacker can extend the lifetime of any issued credential that uses the compromised key for revocation to its maximum validity period; but, the attacker cannot issue new credentials, nor can they affect the revocation of credentials issued after the intrusion has been detected (at which point, a new revocation key is used).

The decision as to which revocation mechanism to use depends on the specifics of the system; in particular, how often are credentials revoked (and for what reason), how stringent the revocation requirements are, what the communication and processing costs and capabilities are, *etc.* For environments where quick revocation is not necessary, time-based expiration may be sufficient; at the other end of the spectrum, a certificate status check protocol may be used to provide near real-time revocation services. (Note however that even Kerberos uses an 8-hour window of revocation, by issuing tickets that are valid for that long, as a tradeoff between efficiency and security.) Luckily, the exact revocation requirements for any particular credential can be encoded in the credential itself; so an administrator's credentials may require an online status check for every use, whereas a user's revocation requirements may be considerably more lax. Furthermore, these requirements can change over time (with each new version of the credential that is issued).

Chapter 5

Implementation of STRONGMAN

This chapter describes the prototype implementation of STRONGMAN. To test our architecture, we converted two different applications such that they were managed by STRONGMAN: IPsec (see Appendix B for a brief tutorial) and the Apache web server. The reasons we picked these are:

- HTTP is in wide use, with a considerable body of work focused on the analysis of its performance under various access scenarios. IPsec is not in as wide use but, as the only widely-accepted network-layer security protocol, it is expected that its use will increase in the next years, especially if the next generation IP protocol[DH96] is widely deployed.
- HTTP and IPsec live in different layers of the network stack, and are good examples of applications which can have useful dependencies.
- It was easy for us to experiment with these protocols, as open-source implementations exist for a variety of operating systems

We built our prototype on the OpenBSD[dRHG$^+$99] platform, since it includes both a full IPsec implementation and the Apache web server.

The first two sections describe our approach to managing IPsec and HTTP respectively through KeyNote credentials. They are followed by a discussion of the "high level" languages used for expressing policy for these applications and the changes needed at the operating system and browser to support the composition primitive. Note that if the applications were to be used in isolation, no changes whatsoever are needed in the operating system (or, more generally, at the interface between any two layers or components). The chapter concludes with an experimental evaluation of STRONGMAN. The reader is referred to Appendix C, for a brief tutorial on IPsec.

5.1 Trust Management for IPsec

We start by examining the security policy issues pertinent to the network layer. We then explain our approach towards addressing these issues[1]

5.1.1 IPsec Policy Issues

When an incoming packet arrives from the network, the host first determines the processing it requires:

- If the packet is not protected, should it be accepted ? This is essentially the "traditional" packet filtering problem, as performed, *e.g.,* by network firewalls.
- If the packet is encapsulated under the security protocol:
  - Is there correct key material (contained in the specified SA) required to decapsulate it ?

[1]Integration of KeyNote and IPsec was joint work with Matt Blaze and John Ioannidis, both at AT&T Research.

  - Should the resulting packet (after decapsulation) be accepted ? A second stage of packet filtering occurs at this point. A packet may be successfully decapsulated and still not be acceptable (*e.g.,* a decapsulated packet with an invalid source address, or a packet attempting delivery to some port not permitted by the receiver's policy).

A security endpoint makes similar decisions when an outgoing packet is ready to be sent:

- Is there a security association (SA) that should be applied to this packet ? If there are several applicable SAs, which one should be selected?

- If there is no SA available, how should the packet be handled ? It may be forwarded to some network interface, dropped, or queued until an SA is made available, possibly after triggering some automated key management mechanism such as IKE, the Internet Key Exchange protocol[HC98].

Observe that because these questions are asked on packet-by-packet basis, packet-based policy filtering must be performed, and any related security transforms applied, quickly enough to keep up with network data rates. This implies that in all but the slowest network environments there is insufficient time to process elaborate security languages, perform public key operations, traverse large tables, or resolve rule conflicts in any sophisticated manner.

IPsec implementations (and most other network-layer entities that enforce security policy, such as firewalls), therefore employ simple, filter-based languages for configuring their packet-handling policies. In general, these languages specify routing rules for handling packets that match bit patterns in packet headers, based on such parameters as incoming and outgoing addresses and ports, services, packet options, *etc.*[MJ93]

IPsec policy control need not be limited to packet filtering, however. A great deal of flexibility is available in the control of when security associations are created and what packet filters are associated with them.

Most commonly however, in current implementations, the IPsec user or administrator is forced to provide “all or nothing” access, in which holders of a set of keys (or those certified by a particular authority) are allowed to create any kind of security association they wish, and others can do nothing at all.

5.1.2 Our Solution

A basic parameter of the packet processing problems mentioned in the previous section is the question of whether a packet falls under the scope of some Security Association (SA). SAs contain and manage the key material required to perform network-layer security protocol transforms. How then, do SAs get created ?

The obvious approach is to trigger the creation of a new SA whenever communication with a new host is attempted, if that attempt would fail the packet-level security policy. The protocol would be based on a public-key or Needham-Schroeder [NS78] scheme.

Unfortunately, protocols that merely arrange for packets to be protected under security associations do nothing to address the problem of enforcing a policy regarding the flow of incoming or outgoing traffic. As is discussed in Appendix B, policy control is a central motivating factor for the use of network-layer security protocols in the first place.

In general, and rather surprisingly, security association policy is largely an open problem – one with very important practical security implications and with the potential to provide a solid framework for analysis of network security properties.

Fortunately, the problem of policy management for security associations can be distinguished in several important ways from the problem of filtering individual packets:

- SAs tend to be rather long-lived; there is locality of reference insofar as hosts that have exchanged one packet are very likely to also exchange others in the near future [Mog92, LS90, CDJM91, Ioa93].
- It is acceptable that policy controls on SA creation should require substantially more resources than could be expended on processing every packet (*e.g.,* public key operations, several packet exchanges, policy evaluation, *etc.*).
- The result of negotiating an SA between two hosts can provide (among other things) parameters for more efficient, lower-level packet policy (filtering) operations. For example, at the end of negotiating an SA, firewall filters can be installed that permit a specific traffic class (*e.g.,* "packets from host A to network B are allowed through the firewall, if they arrive encrypted").

The problem of controlling IPsec SAs is easy to formulate as a trust-management problem: the SA creation process (usually a daemon running IKE) needs to check for compliance whenever an SA is to be created. Here, the actions represent the packet filtering rules required to allow two hosts to conform to each other's higher-level policies.

This leads naturally to a framework for trust management for IPsec:

- Each host has its own KeyNote-specified policy governing SA creation. This policy describes the classes of packets and under what circumstances the host will initiate SA creation with other hosts, and also what types of SAs it is willing to allow other hosts to establish (for example, whether encryption will be used and if so what algorithms are acceptable).

- When two hosts discover that they require an SA, they each propose to the other the "least powerful" packet-filtering rules that would enable them to accomplish their communication objective. Each host sends proposed packet filter rules, along with credentials (certificates) that support the proposal. Any delegation structure between these credentials is entirely implementation dependent, and might include the arbitrary web-of-trust, globally trusted third-parties, such as Certification Authorities (CAs), or anything in between. In this mode, STRONGMAN incurrs no credential distribution overhead over the unmodified IPsec case.

- Each host queries its KeyNote interpreter to determine whether the proposed packet filters comply with local policy and, if they do, creates the SA containing the specified filters.

Other SA properties can also be subject to KeyNote-controlled policy. For example, the SA policy may specify acceptable cryptographic algorithms and key sizes, the lifetime of the SA, logging and accounting requirements.

Our approach is to divide the problem of policy management into two components: packet filtering, based on rules applied to every packet, and trust management, based on negotiating and deciding which of these rules (and related SA properties, as noted above) are trustworthy enough to install.

This distinction makes it possible to perform the per-packet policy operations at high data rates while effectively establishing more sophisticated trust-management-based policy controls over the traffic passing through a security endpoint. Having such controls in place makes it easier to specify security policy for a large network, and makes it especially natural to integrate automated policy distribution mechanisms.

5.1.3 The OpenBSD IPsec Architecture

In this section we examine how the (unmodified) OpenBSD IPsec implementation interacts with the IKE implementation, named `isakmpd`, and how policy decisions are handled and implemented.

Outgoing packets are processed in the `ip_output()` routine. The Security Policy Database (SPD)[2] is consulted, using information retrieved from the packet itself (*e.g.,* source/destination addresses, transport protocol, ports, *etc.*) to determine whether, and what kind of, IPsec processing is required. If no IPsec processing is necessary or if the necessary SAs are available, the appropriate course of action is taken, ultimately resulting in the packet being transmitted. If the SPD indicates that the packet should be protected, but no SAs are available, `isakmpd` is notified to establish the relevant SAs with the remote host (or a security gateway, depending on what the SPD entry specifies). The information passed to `isakmpd` includes the SPD filter rule that matched the packet; this is used in the IKE protocol to propose the packet selectors[3], which describe the classes of packets that are acceptable for transmission over the SA to be established. The same type of processing occurs for incoming packets that are not IPsec-protected, to determine whether they should be admitted; similar to the outgoing case, `isakmpd` may be notified to establish SAs with the remote host.

When an IPsec-protected packet is received, the relevant SA is located using information extracted from the packet and the various protections are peeled off. The packet is then processed as if it had just been received. Note that the resulting, de-IPsec-ed packet may still be subject to local

[2]The SPD is part of all IPsec implementations[KA98c], and is very similar in form to packet filters (and is typically implemented as one). The typical results of an SPD lookup are accept, drop, and "IPsec-needed". In the latter case, more information may be provided, such as what remote peer to establish the SA with, and what level of protection is needed (encryption, authentication).

[3]These are a pair of network prefix and netmask tuples that describe the types of packets that are allowed to use the SA.

policy, as determined by packet filter rules; that is, just because a packet arrived secured does not mean that it should be accepted. We discuss this issue further below.

5.1.4 Adding KeyNote Policy Control

Because of the structure of the OpenBSD IPsec code, we were able to add KeyNote policy control entirely by modifying the `isakmpd` daemon; no modifications to the kernel were required.

Whenever a new IPsec security association is proposed by a remote host (with the IKE protocol), our KeyNote-based `isakmpd` first collects security-related information about the exchange (from its `exchange` and `sa` structures) and creates KeyNote attributes that describe the proposed exchange. These attributes describe what IPsec protocols are present, the encryption/authentication algorithms and parameters, the SA lifetime, time of day, special SA characteristics such as tunneling, PFS, *etc.,* the address of the remote host, and the packet selectors that generate the filters that govern the SA's traffic. All this information is derived from what the remote host proposed to us (or what we proposed to the remote host, depending on who initiated the IKE exchange).

Once passed to KeyNote, these attributes are available for use by policies (and credentials) in determining whether a particular SA is acceptable or not. Recall that the Conditions field of a KeyNote assertion contains an expression that tests the attributes passed with the query. The IPsec KeyNote attributes were chosen to allow reasonably natural, intuitive expression semantics. For example, to check that the IKE exchange is being performed with the peer at IP address 192.168.1.1, a policy would include the test:

```
remote_ike_address == "192.168.001.001"
```

while a policy that allows only the 3DES algorithm would test that

```
esp_enc_alg == "3des"
```

The KeyNote syntax provides the expected composition rules and boolean operators for creating complex expressions that test multiple attributes.

The particular collection of attributes we chose allows a wide range of possible policies. We designed the implementation to make it easy to add other attributes, should that be required by the policies of applications that we failed to anticipate. A partial list of KeyNote attributes for IPsec is contained in Appendix C.

KeyNote policy control contributed only a negligible increase in the code size of the OpenBSD IPsec implementation. To add KeyNote support we had to add about 1000 lines of "glue" code to `isakmpd`. Almost all of this code is related to data structure management and data formatting for communicating with the KeyNote interpreter. For comparison, the rudimentary configuration file-based system that the KeyNote-based scheme replaces took approximately 300 lines of code. The entire original `isakmpd` itself was about 27000 lines of code (not including the cryptographic libraries). The original `isakmpd` and the KeyNote extensions to it are written in the C language. For more details on `isakmpd` itself see [HK00].

5.2 Trust Management for HTTP

Many of the access control issues we discussed in the previous section are relevant to a web server. In the WWW environment the TLS [DA97] (Transport Layer Security) protocol is used, which protects data transmission at the (TCP) connection layer. The information needed to make an access control decision at the web server is similar to that of the IPsec case (source IP address and port, authentication method, data encryption and integrity algorithms, time of day, *etc.*), with the addition of some application-specific details (specific URL request, arguments to posted forms, *etc.*). In the case of the web server, all the mechanisms (data delivery, access control, security protocol) co-exist in the same address space, making the task of modifying the system to support lazy instantiation easier. Furthermore, the TLS protocol has provisions for exchanging public key certificates as part of the authentication step; similar to IPsec, we can use this mechanism to deliver KeyNote credentials without introducing any distribution overheads.

The credentials delivered through TLS are stored along with the rest of the TLS state (again, this is similar to the IPsec concept of the Security Association). In contrast to IPsec however, it is not known in advance what types of requests will be issued by the client; thus, the KeyNote evaluator will have to be invoked for each request that arrives over that TLS connection. We shall see the performance implications of this later in this chapter (to relieve the reader of any anxiety, the overhead imposed by this evaluation is negligible compared to the cost of serving a URL, much less invoking a program as a result of a user submitting an HTML form). Note however that only a small set of policies needs to be examined per request (only those delivered via TLS); the TLS-provided security provides user traffic and, by extension, request isolation.

Defining a new TLS exchange for transferring KeyNote credentials requires modifying the clients (browsers), which is a difficult proposition; on the other hand, most browsers support X.509-based authentication (where X.509 certificates are exchanged over TLS). While these are insufficient by themselves in determining access, they can be used in conjunction with KeyNote credentials to provide access control: the web server has to acquire the relevant KeyNote credentials from the policy repository based on the public key found in the X.509 certificate, and the action authorizer is set to the public key found in the X.509 certificate. The same approach has been adopted in the IPsec implementation, for the same reason.

One remaining issue is how to treat requests that arrive un-encrypted. For these, no real security can be implemented; however, it may still be desirable for the administrator to be able to specify access control rules based only on the source IP address and the URL being requested (since we do not have any information on the identity of the user, and no other security-relevant information is pertinent); see Section 5.3 for sample KeyNote policies. The complication is that, since no authentication step is performed, no KeyNote credentials are delivered to the web server. There are three ways of addressing this issue:

- Establish static rules at the web server. Since the policies do not take into account the users, they may only depend on the URL and the IP address. If these constrains can be concisely expressed (*e.g.,* anyone coming from a particular IP network can ask for any URL in a particular subtree of the web server content hierarchy), then these rules are simply statically installed. Such rules can also cover the common cases (*e.g.,* anyone may be allowed access to the main page of the web server).

- If the number of policies is large (*e.g.,* because there are many combinations of IP address and URL that are valid but which cannot easily be grouped together), the web server can retrieve policies from the policy repository, based on the source IP address of the request and the URL being asked. This can introduce considerable latency in handling the first request but, since the policy can be cached, later requests' performance will be unaffected. The drawback is that the web server has to do some additional processing (communicate with the repository) for each unprotected request. This can easily lead to a denial of service attack, by inducing the web server to communicate with the repository very frequently.

  To protect against that, rate limits can be set on how often the web server can consult with the repository; for multi-threaded servers, processing limits can be placed on the thread that handles un-encrypted requests.

Furthermore, if a denial of service attack is detected, the server can start demanding that all requests be TLS-protected. Luckily, the HTTP protocol allows redirects as a response to a request for a URL; all the web server has to do is redirect the request to a URL that specifies TLS protection (*i.e.,* redirect “http://x.com/a/b/c” URLs to “https://x.com/a/b/c”). Since all browsers support this functionality, it is an attractive method for countering some classes of denial of service attacks.

- Finally, the user can still provide appropriate credentials by submitting them to a special web form on the server. Unfortunately, this requires further (and fairly invasive) modifications of the web browser (the alternative is to request the user to manually upload the relevant credentials, but such an approach would likely cause a revolution among the users). Alternatively, a smart web proxy that uploads the relevant credentials to the web server can be used on the client side. We did not investigate this approach.

We implemented KeyNote access control as a module for the Apache web server[4]. The implementation took less than 800 lines of C code, thanks to the modularity of Apache. The same code could be used in other web servers with minor modifications.

5.3 High Level Policy Languages

For our prototype high-level policy specification for each of the two applications (IPsec and web server access control) we used two simple languages that are very similar in syntax to access

[4]The implementation was written by Todd Miller (millert@openbsd.org); the code has been made available in the OpenBSD distribution since September 2001, and will appear in release 3.0 of the operating system.

```
set credential lifetime to 3 days
permit user ANGELOS if using strong encryption and \
                          going to 192.168.1.0/24
permit usergroup DSL-USERS if using authentication and \
                        coming from LOCALNET and \
                        going to WEBSERVERS
permit usergroup FACULTY except user JMS if \
                        coming from ANYWHERE and \
                        going to central.cis.upenn.edu \
                            accessing ( IMAP or TELNET )
permit user OTHER_ADMINISTRATOR if using strong encryption and \
                        going to COLUMBIA-WEBSERVERS
permit if coming from TRUSTED-SUBNET and going to SMTP-SERVER \
                        accessing SMTP
```

Figure 5.1: A high-level IPsec policy, enforced at the network layer.

control languages used for firewall configuration. Figures 5.1 and 5.7 give sample policies in these languages.

Each of the 5 rules in Figure 5.1 is translated to one or more KeyNote credentials, as shown in Figures 5.2, 5.3, 5.4, 5.5, and 5.6. Similarly, the translations of the web access control rules in Figure 5.7 are given in Figures 5.8 and 5.9.

In our architecture, policy for different network applications can be expressed in various high-level policy languages or systems, each fine-tuned to the particular application. Each such language is processed by a specialized compiler that can take into consideration such information as network topology or a user database and produces a set of KeyNote credentials. At the absolute minimum, such a compiler would need a knowledge of the public keys identifying the users in the system. Other information is necessary on a per-application basis. For example, knowledge of the network topology is typically useful in specifying packet filtering policy; for web access control, the web servers' content structure and directory layout are probably more useful. The proof-of-concept

```
Authorizer: ADMINISTRATOR_PUBLIC_KEY
Licensees: ANGELOS_PUBLIC_KEY
Comment: This credential was issued on 10/19/2001, at 16:10:23
Conditions: app_domain == ``IPsec policy'' &&
            esp_present == ``yes'' &&
            ( esp_enc_alg == ``3des'' ||
              esp_enc_alg == ``AES'' ||
              esp_enc_alg == ``blowfish'' ) &&
            ( esp_auth_alg == ``hmac-sha1'' ||
              esp_auth_alg == ``hmac-md5'' ) &&
            ( local_filter_type == ``IPv4 subnet'' ||
            ( local_filter_type == ``IPv4 range'' ||
            ( local_filter_type == ``IPv4 address'' ) &&
            local_filter_addr_lower >= ``192.168.001.000'' &&
            local_filter_addr_upper <= ``192.168.001.255'' &&
 GMTTimeOfDay <= ``20011022161024'' -> ``strong encryption'';
Signature: PUBLIC_KEY_SIGNATURE_FROM_ADMINISTRATOR
```

Figure 5.2: Translation of the first high-level IPsec rule. Cryptographic keys have been replaced with symbolic names, in the interest of readability.

languages (examples are shown in Figures 5.1 and 5.7) use a template-based mechanism for generating KeyNote credentials. The capabilities offered by the prototype languages are very primitive, but are comparable to what most administrators currently use to configure firewalls or web server access control.

This decoupling of high and low level policy specification permits a more modular and extensible approach, since languages may be replaced, modified, or new ones added without affecting the underlying system.

The second policy rule in Figure 5.7 shows an example of policy composition through the composition hook: the web administrator is willing to accept the connection if the connection is encrypted via IPsec. The translation to the appropriate KeyNote credential is straightforward; the composition hook is considered as just another action environment variable, and its value is retrieved by the web server from the network stack via the `getsockopt()` system call. The IPsec

```
Authorizer: ADMINISTRATOR_PUBLIC_KEY
Licensees: SOTIRIS_PUBLIC_KEY
Comment: This credential was issued on 10/19/2001, at 16:10:23
Conditions: app_domain == ``IPsec policy'' &&
            ( esp_present == ``yes'' ||
              ah_present == ``yes'' ) &&
            ( esp_auth_alg == ``hmac-sha1'' ||
              esp_auth_alg == ``hmac-md5'' ||
              ah_auth_alg == ``hmac-sha1'' ||
              ah_auth_alg == ``hmac-md5'' ) &&
            local_filter_type == ``IPv4 address'' &&
# The webservers are 158.130.6.{101,102,103}
            ( local_filter == ``158.130.006.101 ||
              local_filter == ``158.130.006.102 ||
              local_filter == ``158.130.006.103 ) &&
            local_filter_type == ``IPv4 address'' &&
            remote_filter_type == ``IPv4 address'' &&
# LOCALNET is 158.130.6.0/23
            remote_filter_addr_lower >= ``158.130.006.000'' &&
            remote_filter_addr_upper <= ``158.130.007.255'' &&
 GMTTimeOfDay <= ``20011022161024'' -> ``strong authentication'';
Signature: PUBLIC_KEY_SIGNATURE_FROM_ADMINISTRATOR
```

Figure 5.3: Translation of the second high-level IPsec rule. For each user, a similar credential will be issued (only the Licensees field changes).

```
Authorizer: ADMINISTRATOR_PUBLIC_KEY
Licensees: MBGREEN_PUBLIC_KEY
Comment: This credential was issued on 10/19/2001, at 16:10:23
Conditions: app_domain == ``IPsec policy'' &&
            esp_present == ``yes'' &&
            ( esp_enc_alg == ``3des'' ||
              esp_enc_alg == ``AES'' ||
              esp_enc_alg == ``blowfish'' ) &&
            ( esp_auth_alg == ``hmac-sha1'' ||
              esp_auth_alg == ``hmac-md5'' ) &&
            local_filter_type == ``IPv4 address'' &&
            local_filter == ``158.130.012.002'' &&
            local_filter_proto == ``tcp'' &&
            ( local_filter_port == ``143'' ||
              local_filter_port == ``220'' ||
              local_filter_port == ``23 ) &&
 GMTTimeOfDay <= ``20011022161024'' -> ``true'';
Signature: PUBLIC_KEY_SIGNATURE_FROM_ADMINISTRATOR
```

Figure 5.4: Translation of the third high-level IPsec rule. One of these credentials is issued to each member of group FACULTY, except for user JMS.

```
Authorizer: ADMINISTRATOR_PUBLIC_KEY
Licensees: OTHER_ADMINISTRATOR_PUBLIC_KEY
Comment: This credential was issued on 10/19/2001, at 16:10:23
Conditions: app_domain == ``IPsec policy'' &&
            ( esp_present == ``yes'' ||
              ah_present == ``yes'' ) &&
            ( esp_enc_alg == ``3des'' ||
              esp_enc_alg == ``AES'' ||
              esp_enc_alg == ``blowfish'' ) &&
            ( esp_auth_alg == ``hmac-sha1'' ||
              esp_auth_alg == ``hmac-md5'' ) &&
# Two web servers
            ( local_filter == ``128.059.016.149'' ||
              local_filter == ``128.059.016.150'' ) &&
 GMTTimeOfDay <= ``20011022161024'' -> ``strong encryption'';
Signature: PUBLIC_KEY_SIGNATURE_FROM_ADMINISTRATOR
```

Figure 5.5: Translation of the fourth high-level IPsec rule. This credential allows another administrator to issue credentials to other users.

```
Authorizer: ADMINISTRATOR_PUBLIC_KEY
Licensees: ``IP:128.59.19/22''
Conditions: app_domain == ``Filtering policy'' &&
            remote_filter_addr_upper <= ``158.130.007.255'' &&
            remote_filter_addr_lower >= ``158.130.006.000'' &&
            local_filter == ``158.130.006.012'' &&
            local_filter_port == ``25'' &&
            GMTTimeOfDay <= ``20011022161024'' -> ``true'';
Signature: PUBLIC_KEY_SIGNATURE_FROM_ADMINISTRATOR
```

Figure 5.6: Translation of the fifth high-level IPsec rule. This is an example of a policy which does not refer to a user, but to an IP address. This is a weak form of authentication, and corresponds to traditional firewall rules.

```
allow usergroup JFK-GROUP if file "~angelos/Papers/jfk.ps"
allow user ANGELOS if directory "/confidential" and \
    application IPSEC says ``strong encryption'' and \
                             coming from LOCALNETWORK
```

Figure 5.7: A high-level web access policy, enforced by the web server.

```
Authorizer: WEB_ADMINISTRATOR_PUBLIC_KEY
Licensees: MAB_PUBLIC_KEY
Comment: This credential was issued on 10/19/2001, at 17:00:02
Conditions: app_domain == ``apache'' &&
            method == ``GET'' &&
 uri == ``http://www.cs.columbia.edu/~angelos/Papers/jfk.ps'' &&
   GMTTimeOfDay <= ``20011022170003'' -> ``true'';
SIGNATURE: PUBLIC_KEY_SIGNATURE_FROM_WEB_ADMINISTRATOR
```

Figure 5.8: Translation of the first web access control rule from Figure 5.7. For each member in group JFK-GROUP, one of these credentials will be generated.

```
Authorizer: WEB_ADMINISTRATOR_PUBLIC_KEY
Licensees: ANGELOS_PUBLIC_KEY
Comment: This credential was issued on 10/19/2001, at 17:00:02
Conditions: app_domain == ``apache'' &&
            method == ``GET'' &&
   uri ~= ``^http://www.cs.columbia.edu/confidential/.*'' &&
   source_address_lower >= ``158.130.006.000'' &&
   source_address_upper <= ``158.130.007.255'' &&
   ipsec_result == ``strong encryption'' &&
   GMTTimeOfDay <= ``20011022170003'' -> ``true'';
SIGNATURE: PUBLIC_KEY_SIGNATURE_FROM_WEB_ADMINISTRATOR
```

Figure 5.9: Translation of the second web access control rule.

administrator generates this value simply by generating a policy rule that specifies strong encryption (as seen in Figures 5.1 and 5.2). In this, the ability of KeyNote to operate in multi-value logic (instead of producing just binary true/false decisions) was particularly useful; the various composition hooks are simply converted into a partially-ordered list of return values, and the KeyNote evaluator returns the highest authorized result when making a decision. This result is stored by IPsec and is later retrieved by the web server, as already discussed.

The only coordination necessary between administrators is an agreement on (or understanding of) the meaning of the values of the composition hooks emitted by the various policies. This agreement maps human-level trust into our policies; in our example, the web server administrator trusts the network security administrator to make reasonable assumptions as to what constitutes "strong encryption". This is also an example of human-level delegation of responsibilities (we expect that the network security administrator should know more than the web server administrator as to what constitutes "strong encryption", at least in the context of network layer security); distilling all this information into the composition hook allows us to hide the details of other applications from the administrator. The administrator does not have to understand the specifics of other security protocols; it is sufficient to know the services these protocols offer and how these map into particular values for the composition hook.

5.4 Operating System Modifications

Finally, recall that to implement policy composition (a more accurate term for our system would be *policy coordination*), we need to provide a signaling mechanism between the security mechanisms that can interact. For example, information from the IPsec stack needs to propagate to the web server, so that the policies we saw in Section 5.3 can in fact interact. In our prototype system, we extended the information available through the Unix *getsockopt()* system call, which gives the caller information about a network connection (more accurately, a network socket), to also return the composition hook from the IPsec layer. The necessary code to do this was on the order of 50 lines of C, since we simply used existing data structures and APIs to store and propagate the information.

As we mentioned in Chapter 4, one drawback of this approach is that the APIs have to be extended to accommodate this information flow. Fortunately, new protocols and services are not introduced very often, and extending the relevant APIs can be done at the time of deployment. Furthermore, the information flow typically is unidirectional, since the sequence of operations is well-defined (*e.g.,* a packet has to be processed by IPsec before it can be passed on to the web server); thus, in some cases, the API developed for communicating between one pair of applications can be used unmodified to support other applications (*e.g.,* the same API, without any additional code, can be used by SSH, telnet, or any other user-level process that needs to predicate its access control decisions on information available from lower levels).

5.5 Performance

Our discussion so far has been largely qualitative. To support our scalability claims, we measured various aspects of system performance, that could potentially lead to performance bottlenecks. While a true test of the scalability characteristics of our system would be actual deployment in a large-scale network, such an experiment is beyond our means. Therefore, our methodology is as follows:

- First, identify the potential bottlenecks in our system, and examine whether they apply to our architecture and, if so, how they can be avoided.
- Based on our architecture, determine how our prototype implementation should behave and perform if it were deployed in a real network.
- Perform a set of experiments in a scale that is within our capabilities in terms of equipment and traffic volumes, and verify that our system follows our predictions.
- Finally, determine how one would validate our results in a truly large-scale network.

Based on our architectural description, potential points of bottleneck are the credential generation (since it requires "heavyweight" digital signature operations), credential distribution, policy evaluation and enforcement, and credential revocation. We start by examining these areas in turn.

5.5.1 Credential Generation

During the policy translation step, a large number of credentials will need to be generated. Although it is difficult to estimate what that number will be for a real system, we can provide an upper bound: assuming a network of N users and M enforcement points, we can expect a worst case scenario of $N \times M$ credentials that need to be generated (when all users are given access to all enforcement points, each user is given a different credential for each enforcement point because they have different access rights in each different enforcement point).

For example, for a network of one million users and 100,000 enforcement points, we need 10^{11} credentials, which implies the same number of digital signature operations. On a lightly-loaded workstation with an 850Mhz Pentium III processor, it is possible to do 100 RSA signature operations per second, for 1024-bit keys (as measured with the OpenSSL cryptographic library, version 0.9.6b, using the "openssl speed rsa" test on an OpenBSD system). To handle 10^{11} credentials on this single workstation, we would need approximately 11574 days (close to 32 years) of continuous credential generation. Luckily, a number of factors help us:

- Given that administration in our system is hierarchical, the task of generating the credentials is distributed among the various administrators. If we assume one administrator for every 1000 enforcement points (for a total of 100 administrators, each having to generate 10^9 credentials), the time it takes to generate this many credentials drops to approximately 115 days.

- Each administrator need not confine herself to one workstation generating credentials: assuming 10 systems per administrator, the credential generation time drops to a little over 11 days.

- Better yet, we can use cryptographic hardware accelerators. Those available in the market (in the form of PCI cards) at the time this thesis was written claim a performance of approximately 1000 operations per second. The average PC has 5 PCI slots available for such cards (plus one for a network adaptor); thus, each PC would be able to generate 5000 credentials per second. If each administrator has one of these systems, credential generation time drops to 2.3 days; with 10 of these systems per administrator, credentials can be generated in less than 6 hours. The cost of this solution is modest: each of these cards costs less than $500 (in late-2001 prices).

- One need not generate this many independent credentials: since KeyNote allows many different rules to be combined in the same credential (thus having fewer but larger credentials), administrators can determine the amount of processing needed. For example, administrators can issue one credential per user for all the enforcement points they are responsible for. In our example, this would reduce the number of credentials that need to be generated from 10^{11} to 10^6, for a total generation time of 5 minutes if the administrator uses one workstation with 5 cryptographic accelerator cards. However, the credentials produced may be large (and thus increase distribution and processing overheads). Thus, a tradeoff between credential generation and processing time exists; each rule added to a KeyNote credential adds approximately 2 microseconds to the evaluation of the credential. By aggregating 10 rules in the same credential (thereby creating a credential that can be used in 10 different enforcement points), we can reduce the amount of time it takes to create the complete credential set to 36 minutes, at the cost of increasing the processing time of the assertion by about 20 microseconds (effectively doubling the evaluation overhead, as we shall see in Section 5.5.3).

- Finally, Moore's law is in our favor: every 18 months, the performance of processors doubles, and can thus handle the increased load more easily. Of course, such gains may be partially offset by the use of longer cryptographic keys for increased security.

Of course, it is unlikely that all users need access to all enforcement points; while we do not have numbers from a large-scale deployed system to determine what is a realistic relation, we would expect the number of credentials in a real system to be significantly smaller.

In our prototype system, the cost of digital signatures dominated the cost of processing the high-level language specification and generating the credentials themselves (minus the digital signature), so we expect the cost of processing the high-level policy specification to scale well with the number of policies, users, and enforcement nodes. Clearly, this depends on the details of the high-level policy specification system itself and is beyond the scope of this thesis.

Finally, notice that in our system, adding a new user only causes generation of that user's credentials; no master policy database needs to be updated or synchronized with. The cost of issuing the new user's credentials is directly proportional to the privileges that user needs.

5.5.2 Credential Distribution

In the case of credential distribution costs, we need to distinguish between two cases:

1. The credentials are stored in a (replicated) repository, and users periodically download them. The credentials are then distributed to the enforcement points through the security protocols employed to communicate with these, as we saw in the case of IPsec (Section 5.1.1) and HTTP (Section 5.2). In this case, the overhead of credential distribution in the actual communications is zero, assuming that a small number of credentials are needed for each transaction. Assuming each user is issued one credential per enforcement point they need to access, and the administrative hierarchy is a tree with average depth D, only D credentials need to be provided to the enforcement point at each authentication transaction. For an administrative hierarchy with 10 layers of administration and an average credential size of

1KB, the authentication protocol would need to deliver 10KB worth of credentials. By comparison, current IKE exchanges (with one X.509 certificate used for authentication) involve approximately 1500 to 2000 bytes of traffic in each direction, although this assumes a "flat" hierarchy; in a comparable hierarchy, this traffic would increase to about 5000 bytes.

However, the policy repository must be able to handle the volume of credential downloads. Notice that since policy is expressed in term of credentials, which are integrity-protected through the digital signature by the administrator, downloading the credentials need not be more involved than the equivalent of an (unprotected) file download (or URL-fetch, if using a web interface for downloading the credentials). No cryptographic protocol is required to protect the policy acquisition, which both simplifies system design and allows the repository to be simpler and be able to handle more requests than otherwise[5]. Furthermore, by using a replicated repository, we can distribute the load and improve system availability.

As an indication, using the example worst case scenario of $N \times M$ credentials from the previous section, and using a replicated repository with 100 nodes, each node must be able to handle 10,000 users, each downloading 100,000 credentials every time a new set of credentials is issued. Assuming an average size of 1KB per credential, each user needs to download 100MB worth of credentials.

Again, credential aggregation can help us somewhat, since the size of the Authorizer, Licensees, and Signature fields in an 1KB credential (assuming 1024-bit RSA keys) would come to approximately 50% of the credential size. In the extreme case where one credential

[5]Since no authentication and access control is done on credential acquisition, any user can see anyone else's credentials, potentially raising privacy issues. One way of addressing this issue is by encrypting the users' credentials with their public keys. This ensures that only the intended users can decrypt the credentials, without requiring an authentication protocol. Under this scheme, however, the user *must* acquire the credentials and pass them on to the enforcement points, since only the user can decrypt her credentials.

is issued per user (containing all the access control rules for that user with respect to all 100,000 enforcement points), that credential would be 50MB, which gives us a lower bound on the amount of information the user has to acquire. Of course, a credential of that size would have a direct adverse impact on all authentication transactions.

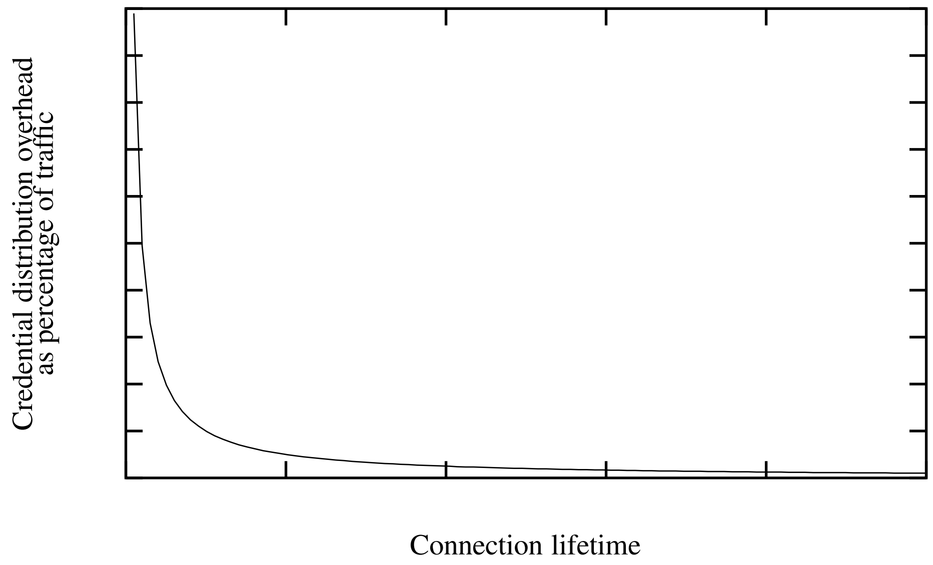
A better approach is to have the user acquire credentials "as needed" from the repository. This introduces some latency in the first authentication exchange done between the user's system and a particular enforcement point, since the user first has to contact the policy repository. Again, the request is the equivalent of an HTTP URL-fetch, and thus we can examine web server performance measurements as an indication of the latency we can expect. Table 5.1 is taken from [BC99] and shows the overhead of downloading a 1KB and a 20KB file over the four combinations of light/heavy network and server load. These file sizes correspond to one credential and a "bundle" of credentials (all credentials starting from the top-level administrator tracing down to the user). The scaling characteristics of the repository can be improved by traditional load balancing techniques employed for improving web performance (caching, higher level of replication of content, content-delivery networks, *etc.*). It should be pointed out that the costs shown in Table 5.1 are incurred once for every *new* enforcement point a client contacts; the credentials can be cached by the client and reused in the future.

2. The other distribution method we examined was that of the enforcement point retrieving the credentials from the repository on behalf of the end user. The exact same overheads we examined in the previous case apply here as well. The attractiveness of this approach is that the clients need not be modified at all; the disadvantage is that the work load of the enforcement point is increased.

Request characteristics	Requests/second	Mean Latency (seconds)
1KB file, light network/server load	4696	0.276
1KB file, light network, heavy server load	768	1.256
1KB file, heavy network, light server load	1543	0.728
1KB file, heavy network/server load	610	1.598
20KB file, light network/server load	1521	0.678
20KB file, light network, heavy server load	572	1.662
20KB file, heavy network, light server load	656	1.503
20KB file, heavy network/server load	444	2.127

Table 5.1: Number of requests per second a web server can handle for two file sizes and under different conditions of network and server load. Also, the corresponding mean latency for downloading these files, as measured by the client. This is the credential distribution cost when either the user or the enforcement point retrieve the credentials from a policy repository on demand ("lazy instantiation"). The measurements were taken with a 333Mhz Pentium Pro acting as a web server on a 10Mbps ethernet. Faster processors and network speeds would improve both the number of requests and latency measured; counter-balancing this, client systems will typically have to contact the policy repository over a wide area network.

The question is whether this overhead is acceptable. As has been previously shown in [Mog92, LS90, CDJM91, Ioa93], network traffic exhibits high spatial and temporal locality of reference (there is a period of time where a host communicates with a small number of other hosts/servers, and then it switches to another set of hosts/servers with which it exchanges traffic). The higher this locality of reference, the more acceptable this distribution overhead is (since it is amortized over relatively large amounts of traffic, for relatively longer periods of time). Although the precise characteristics (*e.g.,* the scale) of a curve describing the overhead in relation to the traffic is not known, it would be roughly described by the formula O/T, where O is the initial overhead and T is the period of time the credential retrieved are useful for. The following graph gives the general shape of the curve. Where a particular network lies on the graph depends on the actual traffic that is carried. We expect that the stronger the locality of reference a network exhibits, the smaller the credential distribution overhead will be (we move to the right in the graph).



Comparing the credential distribution overhead with the elapsed time for performing an IPsec key negotiation and authentication exchange (using the IKE protocol) shows us that the overhead is between 6% and 30% of the full IKE negotiation (a full IKE negotiation on the local network takes approximately 4 seconds to complete). Again, this overhead is incurred only the first time a pair of nodes communicate; the credentials are cached and reused thereafter.

5.5.3 Policy Evaluation and Enforcement

The overhead of KeyNote in the IKE exchanges is negligible compared to the cost of performing public-key operations. Assertion evaluation (without any cryptographic verification) is approximately 120 microseconds on a Pentium II processor running at 450Mhz. With a small number of assertions[6] (less than 15), the initialization and verification overhead is approximately 130 microseconds. This number increases linearly with the size and the number of assertions that are actually evaluated, each such assertion adding approximately 20 microseconds. The bulk of the

[6]This is an artifact of the prototype implementation, which uses a small hash table (of 37 entries) to store credentials. Since in most cases we do not need to worry about more credentials associated with a request, this is sufficient for most compliance checkers. If larger credential sets are needed, the size of the hash table can be increased accordingly.

initialization cost (100 microseconds) relates to data marshaling before it is passed on to KeyNote. Compared with the time it takes to complete a IKE exchange (approximately 4 seconds on the local network), this cost is negligible.

To this, we have to add the cost of signature verification for the credentials. A workstation with a Pentium III processor running at 850Mhz can perform 1,400 RSA signature verification operations per second[7]. Thus, the overhead of verifying the signature of 10 KeyNote credentials comes to 7 milliseconds. While this overhead dominates the cost of the policy evaluation itself, it is still a few orders of magnitude smaller than the cost of the IKE exchange itself. For extremely loaded enforcement points, cryptographic hardware accelerators can be used to increase the number of signatures that can be verified, and thus increase the throughput of the access control mechanism. The same observations hold for the use of KeyNote credentials and policy evaluation in the web server access control scenario.

Furthermore, notice that we have used the KeyNote credentials as part of the authentication phase. The same costs with respect to digital signatures would be incurred with any other public-key-based authentication scheme, *e.g.,* when using X.509 certificates for authentication.

Finally, the signature verification overhead is incurred only the first time a credential is used in evaluating a request; thereafter, the signature need not be verified again, as long as the credential is cached by the IKE daemon or the web server.

[7]Signature verifications are significantly cheaper than signature computations themselves, because typical RSA implementations use a small public exponent. In the OpenSSL library, the default exponent is $2^{16} + 1$, which is 17 bits long; the private exponent, which is used for the signature computation, is significantly larger (close to 1024 bits).

5.5.4 Credential Revocation

The cost of credential revocation depends on the particular mechanism that is used, how often credentials are revoked in the real system, as well as how small we wish to make the window of vulnerability (the elapsed time between privilege revocation by the administrator and when that revocation has propagated throughout the network).

In Section 4.2, we discussed the different revocation mechanisms. We shall now examine the overhead they impose to our architecture.

- One approach is to depend on the validity period encoded in the credential itself; after the credential has expired, it will not be contributing to any policy decisions. If we wish to make the window of vulnerability short, we need to issue credentials with correspondingly short lifetimes. This requires that all credentials be re-generated for all users at least this often; furthermore, users or enforcement points need to re-acquire those credential as often.

  In our discussion in Sections 5.5.1 and 5.5.2, we saw the overheads involved with re-generating the complete credential set and their distribution. A realistic credential lifetime could be on the order of one day, assuming the credential distribution mechanism can support this (for comparison, Kerberos uses a default ticket lifetime of 8 hours). This lifetime is sufficient to amortize the credential distribution costs over large amounts of traffic. A small lifetime for the credentials means that the policy has to be re-instantiated relatively often; on the other hand, prudent cryptographic practices suggest that the authentication context between any two communicating parties should be re-created (or re-validated) periodically, and that is what many cryptographic protocols do (IPsec and TLS included).

The drawback of this mechanism is that, the shorter the credential lifetime, the more communication is needed between the repository and the users and enforcement points. For a given system, the absolute minimal lifetime for a credential is the rate at which the credential set can be re-created. In the example earlier in this section, this can be as short as 6 hours or as long as 32 years.

- If we wish to limit the window of vulnerability to an interval shorter than the credential regeneration cycle, we can issue long-lived credentials which require some proof of validity. We can use any of the following three mechanisms to that end:

    1. Certificate revocation lists (CRLs) can be used when relatively few credentials are revoked at any particular time. How often these are issued dictates the window of vulnerability. The advantage of this approach is that the amount of work a CRL issuer needs to do is much less than re-generating the complete credential set. Certificates are removed from the CRL when their lifetime has expired; this mechanism ensures that CRLs do not grow monotonically. If only a small number of credentials are revoked in any particular period, the cost of distributing the CRL to users or enforcement points is similar to that of distributing a credential. Enforcement points can provide the CRL to users and enforcement points when the credentials themselves are downloaded. If the credentials are used after the lifetime of the CRL itself, a new CRL needs to be downloaded. Since every administrator needs to issue their own CRL, the overhead of contacting this many CRL issuers can quickly add up to a significant fraction of the authentication exchange (for 10 levels of administration, the overhead would be 5 times the elapsed time of an IKE exchange). To prevent this, CRL issuers can poll their

parents in the hierarchy periodically and retrieve the fresh CRL; the whole CRL bundle is then made available to the user or enforcement point.

The cost of this approach from the point of view of the enforcement point is that of verifying as many additional signatures as credentials that are needed for a policy evaluation. Since signature verification is relatively cheap (compared to a full-blown authentication protocol exchange), this is not a big consideration. From the point of view of the user, there is no additional cost. Finally, the administrator is generating the CRL periodically (at the window of vulnerability interval), but does not need to re-generate the credential set. If the CRL grows large, the distribution and processing cost for the enforcement point increase at the same rate. The drawback is that the CRL needs to be generated even if no credentials are revoked.

2. Refresher credentials are issued when a user or enforcement point requests them. They are used to prove that a particular credential is still valid. By making the lifetime of the refresher credentials short, we have the same overheads as in the case of complete credential-set re-generation, with two advantages: first, the key used to sign the user's credential and the key used to sign the refresher credential need not be the same[8]. Thus, compromise of the refresher credential issuer host allows the attacker to extend the lifetime of already-issued credentials, but not issue new credentials. The key used to sign the actual credentials in this case can be kept offline, or in a considerably more "hardened" system than one with which frequent user interaction is needed. Compared with Kerberos, where the KDC and TGS keys have to be kept online and be involved

[8]This is in fact true for all revocation schemes except simple lifetime expiration, but the comparison is particularly relevant in the case of refresher credentials.

in all authentication exchanges, this approach exhibits better fault-tolerance (intrusion-tolerance) behavior.

Secondly, rather than issuing one refresher credential per user credential, we can issue refresher credentials that cover multiple user credentials, as shown in Figure 5.10, reducing the number of credentials that need to be re-issued compared with the short-lived credential approach above. For example, by issuing refresher credentials that cover 10 user credentials, we reduce the issuing requirements by one order of magnitude. On the other hand, the number of credentials that need to be evaluated by the enforcement point has tripled, with a corresponding increase in the cost of signature verifications and policy evaluation.

As a further data point, we can attempt a crude comparison with a Kerberos KDC[9]: assuming a Kerberos installation and a refresher credential issuer that handle the same number of users, issuing (respectively) tickets and refresher credentials of the same lifetime (the Kerberos default is 8 hours). To the author's knowledge, Kerberos KDCs do not use cryptographic accelerators; assuming the 850Mhz Pentium III workstation as the KDC, approximately 2,500 tickets can be issued per second. The same workstation acting as a refresher credential issuer cannot handle the same volume using software-only RSA signatures, as we saw in Section 5.5.3. However, using hardware cryptographic accelerators, it can handle almost twice as many refresher credentials per second. Of course, a Kerberos system with cryptographic accelerators would be able to handle many more tickets as well[10]. However, this comparison tells us that one such

[9]To my surprise, there do not seem to be any published results on Kerberos performance when deployed on a large network.

[10]The exact number depends, among other factors, on how key-agile the hardware is: that is, how fast can it encrypt using 3DES when the key changes for each encryption. Typical hardware accelerators' performance drops by a factor

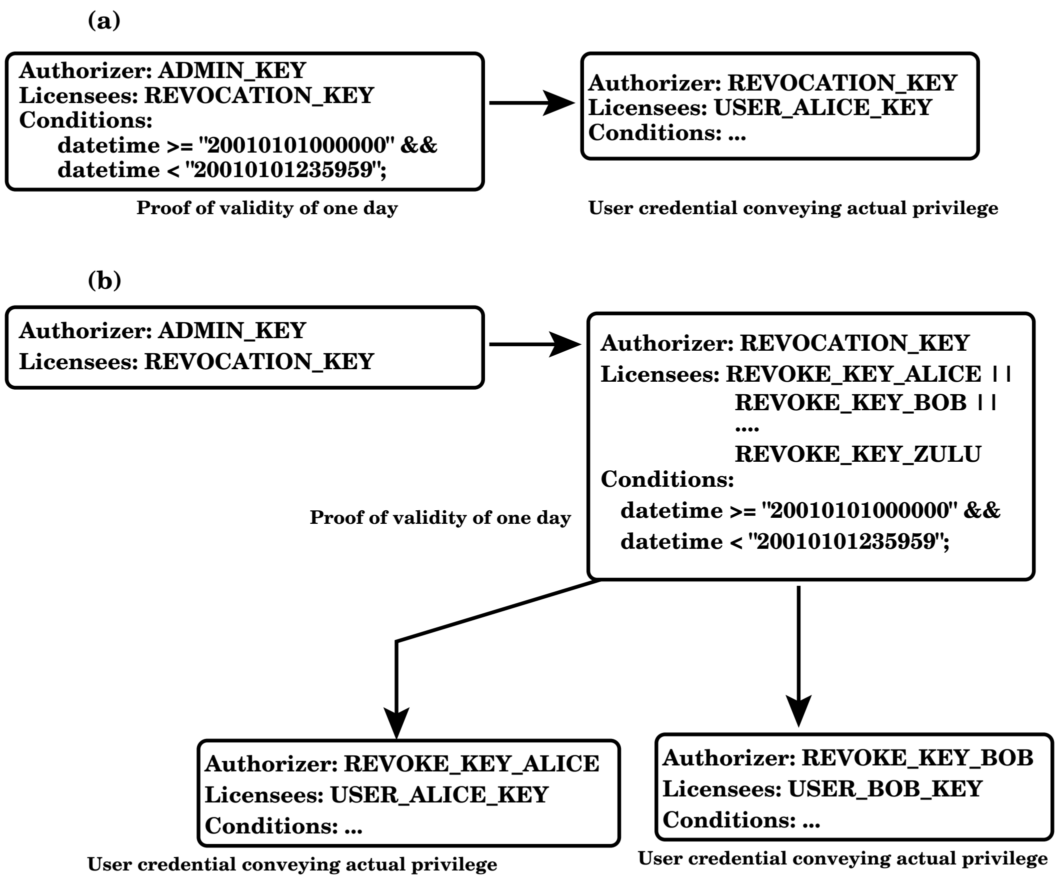

(a)
Authorizer: ADMIN_KEY
Licensees: REVOCATION_KEY
Conditions:
datetime >= "20010101000000" &&
datetime < "20010101235959";
Proof of validity of one day
Authorizer: REVOCATION_KEY
Licensees: USER_ALICE_KEY
Conditions: ...
User credential conveying actual privilege
(b)
Authorizer: ADMIN_KEY
Licensees: REVOCATION_KEY
Authorizer: REVOCATION_KEY
Licensees: REVOKE_KEY_ALICE | |
REVOKE_KEY_BOB | |
....
REVOKE_KEY_ZULU
Conditions:
datetime >= "20010101000000" &&
datetime < "20010101235959";
Proof of validity of one day
Authorizer: REVOKE_KEY_ALICE
Licensees: USER_ALICE_KEY
Conditions: ...
User credential conveying actual privilege
Authorizer: REVOKE_KEY_BOB
Licensees: USER_BOB_KEY
Conditions: ...
User credential conveying actual privilege

Figure 5.10: Proof of validity in the form of KeyNote credentials that delegate to the actual user, shown in (a). This approach requires no changes in the compliance checking mechanism or credential distribution. Furthermore, by using a proof of validity that applies to large numbers of user credentials simultaneously, as shown in (b), we can reduce the number of credentials that need to be periodically re-issued, compared with the same number in the case of short-lived credentials.

system should easily be able to handle the same load as existing Kerberos installations. Since we can distribute the task of issuing refresher credentials among several servers, this part of our system can scale well.

3. Using an Online Certificate Status Check (OCSP) protocol requires the enforcement

of 2 to 5 when the key changes often. On the other hand, a credential refresher issuer would be using the same RSA key for all signature operations, thus avoiding any such penalty.

point to contact the credential issuer (or other authorized entity) to determine the validity of a credential every time that credential is used. This approach allows for the shortest possible window of vulnerability, but has the drawback of significantly increasing the work load of both the enforcement points and the credential issuers (both in terms of network traffic, and thus latency per policy evaluation, and in terms of processing, since OCSP requires that at least one public key operation be done at the OCSP server). Assuming that the cost of doing an OCSP exchange is the same as the heavy network/web server load from Table 5.1, use of OCSP will increase policy evaluation overhead at the enforcement point by 2 seconds, if the various credentials are verified in parallel. Furthermore, for every policy evaluation anywhere in the network, there are D OCSP protocol runs (where D is the average number of credentials used in a policy evaluation).

Some versions of OCSP allow for a validity period to be returned to the enforcement point; within that validity period, the enforcement point does not need to check the status of the credential again. This approach then becomes similar to the refresher credentials above. The disadvantage is that the OCSP result can only be used at the enforcement point that did the check; if the same credential is used at another enforcement point, that enforcement point needs to run the OCSP protocol itself. Given the locality of reference exhibited by network traffic, we expect that this would not cause significant problems.

Certificate revocation is an area of open research, and no individual approach seems suitable for all possible scenarios. In particular, there has not been any large-scale deployment of a PKI such that we can gain some insights as to the frequency of revocation. Some previous work has drawn a parallel between the frequency of password changes in a Kerberos system and the potential rate of revocation of certificates[MJ00], but the two models are probably not similar.

We should note that use of the various revocation mechanisms can be combined in a single system, to provide different windows of vulnerability for different classes of users. For example, administrator keys may be deemed very valuable, and so an online certificate status check may be required with every use[11]; on the other hand, the consequences of a specific user's key being compromised may be deemed small enough that a lightweight revocation mechanism is used (*e.g.,* a CRL with a long period). The different revocation strategies for the different users can be encoded in the credentials issued for these users.

5.5.5 Lazy Policy Instantiation

The discussion so far has focused on the overheads of the various components of our architecture. While we have made some qualitative arguments with respect to the benefits of our architecture (administrative scalability and increased access control flexibility in comparison to any other system), we need to validate our claim that our architecture allows the enforcement mechanisms to scale. The relevant characteristic of our architecture is the concept of lazy policy instantiation. To examine the performance characteristics of this mechanism, let us examine a simple model we can use to compare our architecture with traditional access control approaches.

5.5.5.1 Policy Scalability Evaluation Model

Assume a network firewall that must enforce N policy rules. Furthermore, let $P(N)$ be the probability distribution describing the number of rules that must be evaluated on a firewall with N

[11]Unfortunately, since such credentials are used by many users, this could create high traffic volumes to the status verification servers.

rules. The distribution depends on the traffic characteristics (what types of packets are seen by the firewall, and how these "match" installed policy rules) as well as the particular enforcement mechanism, *e.g.,* linear list, as is the case with the IPF packet filtering package. (In other approaches, *e.g.,* radix trie[Skl93] or rope-search [WVTP97], the lookup cost does not depend on the number of installed rules; however, the cost there is in terms of increased memory requirements, as we shall see shortly.)

For example, assuming linear filter processing and a uniform distribution of traffic among the policy rules, then the expected cost of processing a single packet, $E(P(N))$, will be $N/2$, *i.e.,* for each packet that arrives at the firewall, on the average half the rules will have to be examined before a decision is made.

Let M be the number of enforcement points. Assume some form of load balancing which tries to get $1/M$ of the packets to go to each enforcement point, and which segregates "flows" (the set of packets covered by a particular policy rule), *e.g.,* a particular TCP connection. Thus, a packet flow is seen by a specific enforcement point. Such load balancing will be imperfect but, in general, the larger the degree of replication is (equivalently, the larger M is), the fewer rules will be active at each enforcement point. If the load balancing is perfect then, if a particular traffic trace over some period of time T matches a total of R rules (where $R <= N$), there will be R/M active rules at each enforcement point. In practice, the load balancing will not be perfect, and some rules will "fire" in more than one enforcement point, so that the average number of active rules per enforcement point will be greater that R/M.

The larger the time interval, T, over which we measure the number of active rules on a given enforcement point, the lower the per-packet cost of dynamically loading and unloading rules will be. At the same time, the number of rules that were active during that interval will also increase, increasing the total cost of loading/unloading rules. If flows are long-lived, then T can be large without increasing the number of active rules in the enforcement point. If flows are relatively short, then the number of active rules will grow rapidly with an increase in T. The lifetime of flows is a function of the traffic distribution.

Let $C(N, M, T)$ represent the mean number of rules active in a single enforcement point at any one time. Clearly, the best case for lazy policy instantiation is $C(N, M, T) = R/M$; this represents the case of perfect load balancing. The worst possible case is $C(N, M, T) = R$. We know that, as T becomes larger, $C(N, M, T)$ behaves as a non-monotonically increasing function (*i.e.,* it cannot shrink). On the other hand, as M grows larger, $C(N, M, T)$ is non-monotonically decreasing (*i.e.,* it cannot grow).

Given these assumptions, the expected per-packet cost of a traditional firewall will be

$$E(P(N)) \times Q$$

where Q is the cost of evaluating a single rule. Furthermore, the space requirements are $O(N)$, regardless of $C(N, M, T)$ (the mean number of active rules). This means that the policy storage-space requirements in a traditional firewall is the same, regardless of the degree of replication (M) and the interval over which we examine traffic (T).

The speed and memory usage complexity of a variety of other lookup algorithms is given in Table 5.2, as reported in [WVTP97]. While the processing complexity of most of these algorithms is independent of the number of policy rules, the memory consumption increases dramatically compared to the simple linear list approach used in most firewalls.

The expected per-packet cost in the case of lazy policy instantiation on a firewall with linear-list filter processing will be

$$E(P(C(N, M, T))) \times Q + (C(N, M, T) \times \textit{(cache load cost + cache unload cost))}\ /T$$

Algorithm	Build	Search	Memory
Binary Search	$O(N \times \log(N))$	$O(\log(2 \times N))$	$O(N)$
Trie	$O(N \times W)$	$O(W)$	$O(N \times W)$
Radix Trie	$O(N \times W)$	$O(W)$	$O(N)$
Basic Scheme	$O(N \times \log(W))$	$O(\log(W))$	$O(N \times \log(W))$
Asymmetric BS	$O(N \times \log(W))$	$O(\log(W))$	$O(N \times \log(W))$
Rope Search	$O(N \times W)$	$O(\log(W))$	$O(N \times \log(W))$
Ternary CAMs	$O(N)$	$O(1)$	$O(N)$

Table 5.2: Speed and memory usage complexity for a variety of fast filter lookup mechanisms. W is the number of bits in the lookup key (*e.g.,* if the lookup mechanism is used for simple IPv4 packet forwarding, the key is an IPv4 address, therefore $W = 32$.)

with space complexity of $O(C(N, M, T)) = O(R/M)$. We expect that for most large-scale systems $R << N$ and $R/M << N$. The cost of dynamically loading and unloading rules from the kernel policy table is essentially that of a system call. As we shall see in Section 5.5.5.2, the cost of loading a new rule in our experimental system is on the order of 10 ms. Unloading rules can cost less, if it is done as a replacement operation.

Finally, notice that for a given $C, P(N), M$, and traffic distribution, we can choose the appropriate T to minimize the per-packet cost of lazy policy instantiation.

Let us look at a specific instance of the model. Consider a network firewall that mediates access to a protected network as the test case of an enforcement point. In a traditional architecture, this firewall has to enforce a set of access control rules that pertain to a set of users and/or remote hosts. For example, consider the fictitious policy shown in Figure 5.11.

These rules are always present at the firewall, and for each legitimate packet that is received, the firewall needs to examine on the average half of them to determine what should happen to the packet, assuming that traffic is uniformly distributed across these rules. If the firewall had to handle 100,000 users (each with their own rule), it would have to examine 50,000 rules on the average. Worse yet, it would have to do this even if only a few users out of the 100,000 were actually

```
permit from any to webserver1 protocol tcp port 80
permit from any to webserver2 protocol tcp port 80
permit from any to webserver3 protocol tcp port 80
permit from any to webserver4 protocol tcp port 80
permit from any to login-host protocol tcp port 23 if encrypted
permit from any to webserver6 protocol tcp port 80 if encrypted
permit from any to webserver7 protocol tcp port 80 if encrypted
permit from any to webserver8 protocol tcp port 80 if encrypted
permit from any to mailserver protocol tcp port 25
permit from any to ima-server protocol tcp port 220 if encrypted
block from any to any
```

Figure 5.11: A simple set of rules a firewall may be called to enforce.

active at any particular time. The workload of the firewall is thus dictated by the number of rules the firewall has to enforce, as opposed to the actual traffic it experiences (the number of users interacting with it). In effect, the firewall is not taking advantage of the statistical multiplexing assumptions on which (most) communication networks are based[12].

Consider the same firewall with the same enforcement mechanism (linear scan of the rules, trying to determine a first match) and the same traffic distribution, but using lazy policy instantiation. This approach does in fact take advantage of statistical multiplexing: the only rules the firewall has to consider are those that correspond to actual traffic. Thus, while the workload is still $N/2$, in this case N is not the number of policies the firewall has to *potentially* enforce, but the number of rules it *actually* has to consider, the value R from the model. This means that the performance curve of the enforcement mechanism follows the performance curve of the network and operating system components.

[12]That is, the fact that the network does not need to be provisioned with enough resources to support the maximum potential utilization (all users accessing it); rather, enough resources are needed to provide service to the active set of users at any particular moment.

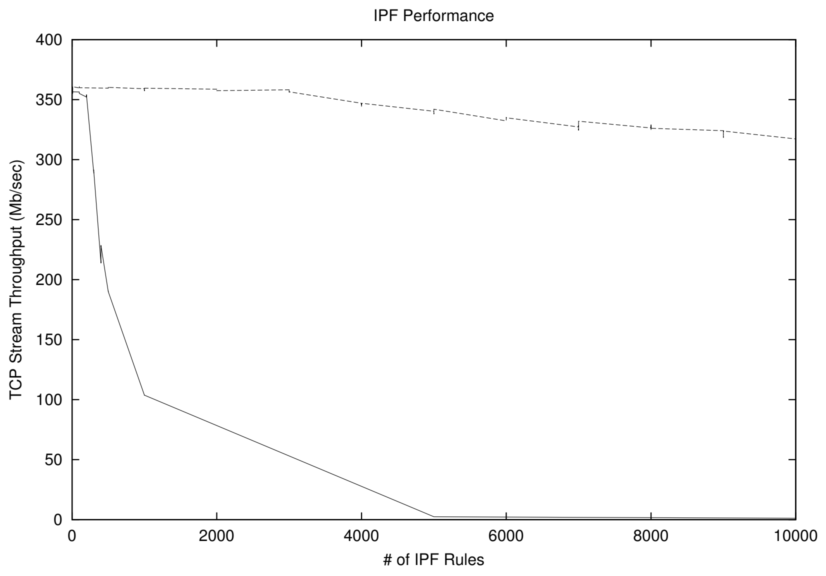


Figure 5.12: TCP performance through a firewall with an increasing number of rules. The two measurements are with IPF and PF respectively, two different packet-filtering packages (the rapidly declining performance was experienced with the initial version of PF in OpenBSD 2.9). On a system with a slower CPU, the decline in throughput is even more pronounced, as there are fewer cycles available per packet. As shown in [Dah95], improvements in network performance will continue to outpace Moore's Law.

No policy	11,131 ms
Lazy instantiation	11,178 ms
IPF	11,151 ms

Table 5.3: 100MB file transfer over TCP, measured in ms.

5.5.5.2 Experimental Evaluation

To validate this conclusion, we constructed a set of experiments. First, we measured the performance of a traditional firewall that has to enforce an increasing number of policies on the same traffic. The experimental setup involved three Pentium III PCs connected in a daisy-chain configuration over dedicated Gigabit ethernet connections, with the middle system enforcing an increasing number of packet filtering rules (from 0 to 10,000 rules). All PCs were running OpenBSD 2.9, using the IPF and PF network packet filtering packages (resulting in two different measurements). In all cases, the traffic was a bulk data transfer over a TCP connection between the other two systems, and the firewall had to go through all the rules in its table before it found the one permitting the TCP traffic through. The performance curves[13] are shown in Figure 5.12. We see that performance drops linearly (or worse) with the number of rules. The same experiment when using lazy instantiation, in which case only one rule would be enforced by the firewall at any single time, gives us constant performance at 360Mbps, irrespective of the number of rules (since only one of these appears in the filter list). As the number of rules that are instantiated at the firewall increases, performance follows the same curve as that shown in Figure 5.12 (depending on the underlying filtering mechanism, IPF or PF). Note however, that the *number* of rules that has to be enforced *and kept in memory* at any time is much smaller than in the traditional firewall case. Thus, the benefit of our architecture depends on the statistical multiplexing properties of the network where our system is deployed.

In the case of a fast lookup mechanism the cost of lookups is independent of the number of total rules. However, the memory storage requirements for these rules increase dramatically: while our test firewall could easily handle 10,000 IPF or PF rules, it refused to accept 10,000 IPsec policy entries, as these occupied considerably more memory. Given that the IPsec rule size ranges from 168 to 312 bits[14], the storage cost in some of the fast schemes increases by a factor of six to eight, as shown in Table 5.2.

Since we do not have access to a large network with many users and firewalls, it is impossible

[13]Our thanks to Stefan Mitchev for instrumenting and performing this experiment.

[14]The low end of the range represents the IPv4 source/destination addresses and netmasks (four quantities of 32 bits each), while the high end of the range represents the IPv6 source/destination addresses (128 bits each) plus two 8-bit prefixes. To each of these size, two 16-bit port numbers and one 8-bit transport protocol number field have to be added.

to know what the actual improvement is. If we had access to one, we could simply monitor the traffic that arrives at a particular firewall over a period of time, and determine the workload of the firewall given a particular set of policies (since we know the actual distribution of traffic over the filtering rules). Furthermore, given that same traffic trace, we could determine how many rules are actually needed at the firewall, and thus determine the workload under those circumstances.

Lazy instantiation allows the system architect to scale the security policy enforcement mechanism at the same rate as the rest of the system. For example, the most common approach to solving scalability problems with a system (*e.g.,* a web server) is by replication and load sharing. Traditional access control systems either do not derive this distribution benefit (because the same access control information is enforced by all the servers), or do so at the expense of availability (by forcing users to access specific servers).

Finally, it should be pointed out that having a decentralized policy evaluation and enforcement system can allow the system architect to eliminate some enforcement points (*e.g.,* a centralized firewall), thus improving overall network efficiency. We instrumented a simple experiment to measure the benefit from such a change in the network infrastructure.

The measurements of Table 5.3 have a server application running on Alice; a client running on Bob connects to Alice and transfers 100MB. The numbers represent the total transfer time of the 100MB file. It is clear that our system does not significantly affect transfer time (the difference is on the order of 0.5%).

In the experiment of Figure 5.13, we used a configuration of four systems (300MHz Pentium II) interconnected via a 100Mbps ethernet hub. One of the four machines is connected to the "outside world" with 100 Mbps ethernet. In the outside world there is an 850 MHz machine (Alice). The "inside" three machines run a simple server accepting connections. The outside machine, through

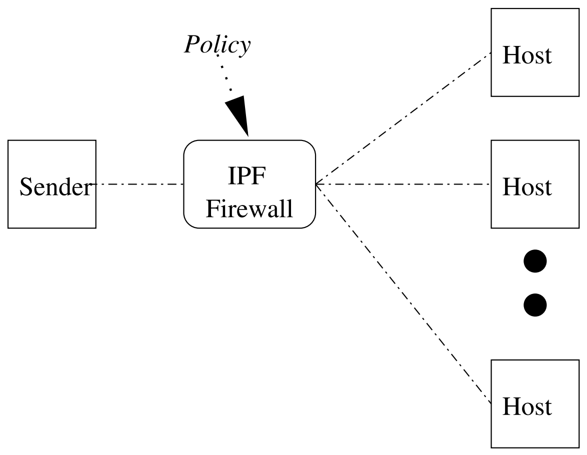


Figure 5.13: Test topology with intermediate firewall.

Insecure	109.1 ms
IPF	134.2ms

Table 5.4: Average connection overhead measured in ms for 100 TCP connections between hosts through a firewall. The additional overhead of IPF filtering is 23%.

the gateway, makes 100 connections in a round robin fashion to the three machines. We measure the time it takes to establish those 100 connections (33 per host). The measurements are given in the table of Figure 5.4.

Using the same end-hosts, we eliminate the gateway machine, with each of the client machines running a KeyNote evaluator and enforcing policy locally (see Figure 5.14). The ethernet hub is connected directly to the outside world; the rest of the configuration remains as in the previous experiment. To test the performance of the firewall in this scenario, we varied the number of hosts that participate in the connection setup. As in the previous experiment we formed 100 connections to the machines running the KeyNote evaluator in a round robin fashion, each time varying the number of participating hosts. We make the assumption that every protected host inside a firewall contributes roughly the same number of rules, and in the classic centralized case the firewall will

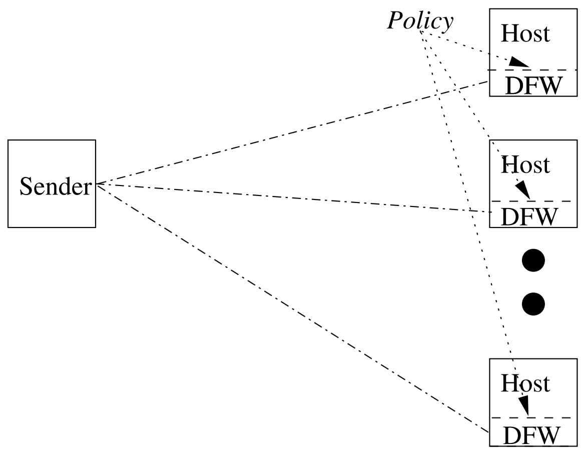


Figure 5.14: Test topology without intermediate firewall.

	1 Host	2 Hosts	3 Hosts
Insecure	56.1 ms	53.1 ms	48.6 ms
Lazy policy instantiation	66.3 ms	58.0 ms	50.5 ms

Table 5.5: Average connection overhead measured in ms for 100 TCP connections spread over one, two and three hosts respectively, using lazy policy instantiation at the end hosts.

have to enforce the union of those rules. Therefore individual machines will have a smaller rule base than a central control point. The measurements and the percentile overheads are given in Tables 5.5 and 5.6 respectively. We have kept the total number of rules constant as in the IPF case, and spread them over an increasing number of machines. This experiment demonstrates the benefit of eliminating intermediate enforcement points, and pushing security functions to the endpoints: a two-fold improvement in performance compared to the centralized approach, in addition to the increased flexibility and scalability offered by our architecture.

The key observation here is that not all users can (or do) access the same enforcement points at the same time; our architecture takes advantage of this fact, by only instantiating rules as-needed at an enforcement point.

	1 Host	2 Hosts	3 Hosts
Overhead	18.2%	9.2%	3.9%

Table 5.6: Processing overhead as the number of hosts increases (and policies are distributed among them). The percentages represent the additional cost of the distributed firewall over the insecure case and are derived from Table 5.5.

Chapter 6

Applications and Future Work

This chapter describes the distributed firewall concept[Bel99, IKBS00], and the prototype implementation. The distributed firewall is one of the different enforcement mechanisms that can be controlled by STRONGMAN; indeed, widespread use of the distributed firewall *requires* a large-scale security management system like STRONGMAN. Conversely, the distributed firewall embodies the philosophy of distributed, fine-grained access control advocated by STRONGMAN.

The second section discusses an intermediate step towards wide-scale deployment of the distributed firewall, the transparent firewall.

6.1 Distributed Firewall

A *firewall* is a collection of components, interposed between two networks, that filters traffic between them according to some security policy [CB94]. Conventional firewalls rely on network topology restrictions to perform this filtering. Furthermore, one key assumption under this model is that everyone on the protected network(s) is trusted (since internal traffic is not seen by the firewall, it cannot be filtered); if that is not the case, then additional internal firewalls have to be deployed in the internal network.

However, as we discussed in Chapter 1, several trends in networking threaten to make traditional firewalls obsolete. In addition to these, traditional firewalls have some other problems:

- Due to the increasing line speeds and the more computation-intensive protocols that a firewall must support (especially IPsec), firewalls tend to become congestion points. This gap between processing and networking speeds is likely to increase, at least for the foreseeable future; while computers (and hence firewalls) are getting faster, the combination of more complex protocols and the tremendous increase in the amount of data that must be passed through the firewall has outpaced, and likely will continue to outpace, Moore's Law [Dah95].

- Large (and even not-so-large) networks today tend to have a large number of entry points (for performance, failover, and other reasons). Furthermore, many sites employ internal firewalls to provide some form of compartmentalization. This makes administration particularly difficult, both from a practical point of view and with regard to policy consistency, since no unified and comprehensive management mechanism exists.

Despite their shortcomings, firewalls are still useful in providing some measure of security. The key reason that firewalls are still useful is that they provide an obvious, mostly hassle-free, mechanism for *enforcing* network security policy. For legacy applications and networks, they are the only mechanism for security. While newer protocols typically (but not always) have some provisions for security, older protocols and their implementations are more difficult, often impossible, to secure. Furthermore, firewalls provide a convenient first-level barrier that allows quick responses to newly-discovered bugs.

To address the shortcomings of firewalls while retaining their advantages, [Bel99] proposed the concept of a *distributed firewall.* In distributed firewalls, security policy is defined centrally but enforced at each individual network endpoint (hosts, routers, *etc.*). The system propagates the central policy to all endpoints. Policy distribution may take various forms. For example, it may be pushed directly to the end systems that have to enforce it, or it may be provided to the users in the form of credentials that they use when trying to communicate with the hosts, or it may be a combination of both. The extent of mutual trust between endpoints is specified by the policy.

To implement a distributed firewall, three components are necessary:

- A language for expressing policies and resolving requests. In their simplest form, policies in a distributed firewall are functionally equivalent to packet filtering rules. However, it is desirable to use an extensible system (so other types of applications and security checks can be specified and enforced in the future). The language and resolution mechanism should also support credentials, for delegation of rights and authentication purposes [BFIK99a].

- A mechanism for safely distributing security policies. This may be the IPsec key management protocol when possible, or some other protocol. The integrity of the policies transferred must be guaranteed, either through the communication protocol or as part of the policy object description (*e.g.,* they may be digitally signed).

- A mechanism that applies the security policy to incoming packets or connections, providing the enforcement part.

Our prototype implementation[1] uses the KeyNote trust-management system, which provides a single, extensible language for expressing policies and credentials. Credentials in KeyNote are

[1]The distributed firewall was joint work with Sotiris Ioannidis and Jonathan M. Smith at the University of Pennsylvania, and Steve Bellovin of AT&T Labs - Research.

signed, thus simple file-transfer protocols may be used for policy distribution. We also make use of the IPsec stack in the OpenBSD system to authenticate users, protect traffic, and distribute credentials. The distribution of credentials and user authentication occurs as part of the Internet Key Exchange (IKE) [HC98] negotiation. Alternatively, policies may be distributed from a central location when a policy update is performed, or they may be fetched as-needed (from a web server, X.500 directory, or through some other protocol).

Since KeyNote allows delegation, decentralized administration becomes feasible (establishing a hierarchy or web of administration, for the different departments or even individual systems). Users are also able to delegate authority to access machines or services they themselves have access to. Although this may initially seem counter-intuitive (after all, firewalls embody the concept of centralized control), in our experience users can almost always bypass a firewall's filtering mechanisms, usually by the most insecure and destructive way possible (*e.g.,* giving away their password, setting up a proxy or login server on some other port, *etc.*). Thus, it is better to allow for some flexibility in the system, as long as the users follow the overall policy. Also note that it is possible to "turn off" delegation.

Thus, the overall security policy relevant to a particular user and a particular end host is the composition of the security policy "pushed" to the end host, any credentials given to the user, and any credentials stored in a central location and retrieved on-demand. Finally, we implement the mechanism that enforces the security policy at a TCP-connection granularity. In our implementation, the mechanism is split in two parts, one residing in the kernel and the other in a user-level process.

6.1.1 Architecture

A distributed firewall such as described in [Bel99], uses a central policy, but pushes enforcement towards the edges. That is, the policy defines what connectivity, inbound and outbound, is permitted; this policy is distributed to all endpoints, which enforce it.

In the full-blown version, endpoints are characterized by their IPsec identity, typically in the form of a certificate. Rather than relying on the topological notions of "inside" and "outside", as is done by a traditional firewall, a distributed firewall assigns certain rights to whichever machines own the private keys corresponding to certain public keys. Thus, the right to connect to the `http` port on a company's internal Web server might be granted to those machines having a certificate name of the form `*.goodfolks.org`, rather than those machines that happen to be connected to an internal wire. A laptop directly connected to the Internet has the same level of protection as does a desktop in the organization's facility. Conversely, a laptop connected to the corporate net by a visitor would not have the proper credentials, and hence would be denied access, even though it is topologically "inside."

To implement a distributed firewall, we need a security policy language that can describe which connections are acceptable, an authentication mechanism, and a policy distribution scheme. As a policy specification language, we use the KeyNote trust-management system.

As an authentication mechanism, we used IPsec for traffic protection and user/host authentication. While we can, in principle, use application-specific security mechanisms (*e.g.,* SSL-enabled web-browsing), this would require extensive modifications of all such applications to make them aware of the filtering mechanism. Furthermore, we would then depend on the good behavior of the very applications we are trying to protect. Finally, it would be impossible to secure legacy applications with inadequate provisioning for security.

When it comes to policy distribution, we have a number of choices:

- We can distribute the KeyNote (or other) credentials to the various end users. The users can then deliver their credentials to the end hosts through the IKE protocol. The users do not have to be online for the policy update; rather, they can periodically retrieve the credentials from a repository (such as a web server). Since the credentials are signed and can be transmitted over an insecure connection, users could retrieve their new credentials even when the old ones have expired. This approach also prevents, or at least mitigates, the effects of some possible denial of service attacks.

- The credentials can be pushed directly to the end hosts, where they would be immediately available to the policy verifier. Since every host would need a large number, if not all, of the credentials for every user, the storage and transmission bandwidth requirements are higher than in the previous case.

- The credentials can be placed in a repository where they can be fetched as needed by the hosts. This requires constant availability of the repository, and may impose some delays in the resolution of request (such as a TCP connection establishment).

While the first case is probably the most attractive from an engineering point of view, not all IKE implementations support distribution of KeyNote credentials. Furthermore, some IPsec implementations do not support connection-grained security. Finally, since IPsec is not (yet) in wide use, it is desirable to allow for a policy-based filtering that does not depend on IPsec. Thus, it is necessary to provide a *policy resolution* mechanism that takes into consideration the connection parameters, the local policies, and any available credentials (retrieved through IPsec or other means), and determines whether the connection should be allowed. We describe our implementation of such a mechanism for the OpenBSD system in Section 6.1.2.

6.1.2 Implementation

For our development platform we use the OpenBSD operating system [dRHG+99]. OpenBSD provides an attractive platform for developing security applications because of the well-integrated security features and libraries (an IPsec stack, SSL, KeyNote, *etc.*). However, similar implementations are possible under other operating systems.

Our system has three components: a set of kernel extensions, which implement the enforcement mechanisms, a user level daemon process, which implements the distributed firewall policies, and a device driver, which is used for two-way communication between the kernel and the policy daemon. Our prototype implementation totals approximately 1150 lines of C code; each component is roughly the same size.

Figure 6.1 shows a graphical representation of the system, with all its components. In the following three subsections we describe the various parts of the architecture, their functionality, and how they interact with each other.

6.1.2.1 Kernel Extensions

For our working prototype we focused our efforts on the control of the TCP connections. Similar principles can be applied to other protocols; for unreliable protocols, some form of reply caching is desirable to improve performance.

In the Unix operating system users create outgoing and allow incoming TCP connections using the `connect(2)` and `accept(2)` system calls respectively. Since any user has access to these

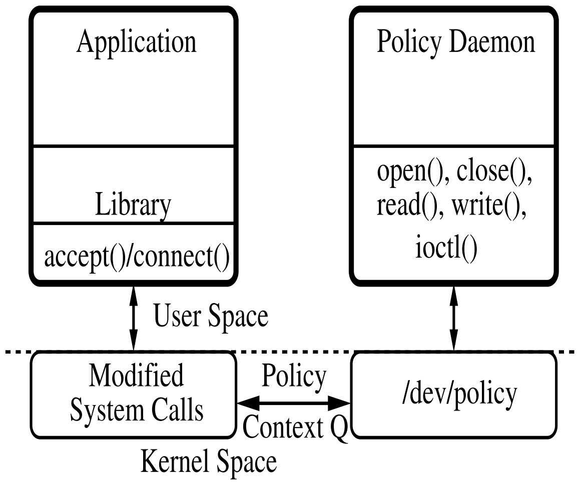


Figure 6.1: The figure shows a graphical representation of the system, with all its components. The core of the enforcement mechanism lives in kernel space and is comprised of the two modified system calls that interest us, `connect(2)` and `accept(2)`. The policy specification and processing unit lives in user space inside the policy daemon process. The two units communicate via a loadable pseudo device driver interface. Messages travel from the system call layer to the user level daemon and back using the *policy context queue*.

system calls, some “filtering” mechanism is needed. This filtering should be based on a policy that is set by the administrator.

Filters can be implemented either in user space or inside the kernel. Each approach has its advantages and disadvantages.

A user level approach, as depicted in Figure 6.2, requires each application of interest to be linked with a library that provides the required security mechanisms, *e.g.,* a modified `libc`. This has the advantage of operating system-independence, and thus does not require any changes to the kernel code. However, such a scheme does not guarantee that the applications *will* use the modified library, potentially leading to a major security problem.

A kernel level approach, as shown in the left side of Figure 6.1, requires modifications to the operating system kernel. This restricts us to open source operating systems like BSD and Linux. The main advantage of this approach is that the additional security mechanisms can be enforced

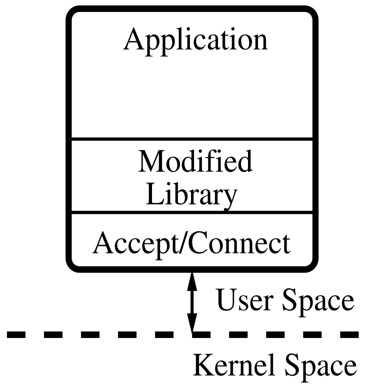


Figure 6.2: Wrappers for filtering the `connect(2)` and `accept(2)` system calls are added to a system library. While this approach offers considerable flexibility, it suffers from its inability to guarantee the enforcement of security policies, as applications might not link with the appropriate library.

transparently on the applications.

As we mentioned previously, the two system calls we need to filter are `connect(2)` and `accept(2)`. When a `connect(2)` is issued by a user application and the call traps into the kernel, we create what we call a *policy context* (see Figure 6.3), associated with that connection.

The policy context is a container for all the information related to that specific connection. We associate a sequence number to each such context and then we start filling it with all the information the *policy daemon* will need to decide whether to permit it or not. In the case of the `connect(2)`, this includes the ID of the user that initiated the connection, the destination address and port, *etc.* Any credentials acquired through IPsec may also be added to the context at this stage. There is no limit as to the kind or amount of information we can associate with a context. We can, for example, include the time of day or the number of other open connections of that user, if we want them to be considered by our decision–making strategy.

Once all the information is in place, we *commit* that context. The commit operation adds the context to the list of contexts the policy daemon needs to handle. After this, the application is blocked waiting for the policy daemon reply.

```
typedef struct  policy_mbuf     policy_mbuf;
struct  policy_mbuf     {
        policy_mbuf     *next;
        int              length;
        char             data[POLICY_DATA_SIZE];
};

typedef struct  policy_context  policy_context;
struct  policy_context  {
        policy_mbuf     *p_mbuf;
        u_int32_t        sequence;
        char            *reply;
        policy_context  *policy_context_next;
};

policy_context  *policy_create_context(void);
void            policy_destroy_context(policy_context *);
void            policy_commit_context(policy_context *);
void            policy_add_int(policy_context *, char *, int);
void            policy_add_string(policy_context *, char *,
                                  char *);
void            policy_add_ipv4addr(policy_context *, char *,
                                    in_addr_t *);
```

Figure 6.3: The `connect(2)` and `accept(2)` system calls create *contexts* which contain information relevant to that connection. These are appended to a queue from which the policy daemon will receive and process them. The policy daemon will then return to the kernel a decision on whether to accept or deny the connection.

Accepting a connection works in a similar fashion. When `accept(2)` enters the kernel, it blocks until an incoming connection request arrives. Upon receipt, we allocate a new context which we fill in similarly to the `connect(2)` case. The only difference is that we now also include the source address and port. The context is then enqueued, and the process blocks waiting for a reply from the policy daemon.

In the next section we discuss how messages are passed between the kernel and the policy daemon.

6.1.2.2 Policy Device

To maximize the flexibility of our system and allow for easy experimentation, we made the policy daemon a user-level process. To support this architecture, we implemented a *pseudo device driver*, `/dev/policy`, that serves as a communication path between the user–space policy daemon, and the modified system calls in the kernel. Our device driver supports the usual operations (`open(2)`, `close(2)`, `read(2)`, `write(2)`, and `ioctl(2)`). Furthermore, we have implemented the device driver as a loadable module. This increases the functionality of our system even more, since we can add functionality dynamically, without needing to recompile the whole kernel.

If no policy daemon has opened `/dev/policy`, no connection filtering is done. Opening the device activates the distributed firewall and initializes data structures. All subsequent `connect(2)` and `accept(2)` calls will go through the procedure described in the previous section. Closing the device will free any allocated resources and disable the distributed firewall.

When reading from the device the policy daemon blocks until there are requests to be served. The policy daemon handles the policy resolution messages from the kernel, and writes back a reply. The `write(2)` is responsible for returning the policy daemons decision to the blocked connection call, and then waking it up. It should be noted that both the device and the associated messaging protocol are not tied to any particular type of application, and may in fact be used without any modifications by other kernel components that require similar security policy handling.

Finally, the kernel and the policy daemon can re–synchronize in case of any errors in creating or parsing the request messages, by discarding the current policy context and dropping the associated connection.

6.1.3 Policy Daemon

The third and last component of our system is the policy daemon. It is a user level process responsible for making decisions, based on policies that are specified by some administrator and credentials retrieved remotely or provided by the kernel, on whether to allow or deny connections.

Policies are initially read in from a file. It is possible to remove old policies and add new ones dynamically. In the current implementation, such policy changes only affect new connections.

Communication between the policy daemon and the kernel is possible, as we mentioned earlier, using the `policy` device. The daemon receives each request (see Figure 6.4) from the kernel by reading the device. The request contains all the information relevant to that connection as described in Section 6.1.2.1. Processing of the request is done by the daemon using the KeyNote library, and a decision to accept or deny it is reached. Finally the daemon writes the reply back to the kernel and waits for the next request. While the information received in a particular message is application-dependent (in our case, relevant to the distributed firewall), the daemon itself has no awareness of the specific application. Thus, it can be used to provide policy resolution services for many different applications, literally without any modifications.

When using a remote repository server, the daemon can fetch a credential based on the ID of the

```
u_int32_t seq;     /* Sequence Number */
u_int32_t uid;             /* User Id */
u_int32_t N;     /* Number of Fields */
u_int32_t l[N]; /* Lengths of Fields */
char      *field[N];       /* Fields */
```

Figure 6.4: The request to the policy daemon is comprised of the following fields: a sequence number uniquely identifying the request, the ID of the user the connection request belongs to, the number of information fields that will be included in the request, the lengths of those fields, and finally the fields themselves.

user associated with a connection, or with the local or remote IP address. A very simple approach to that is fetching the credentials via HTTP from a remote web server. The credentials are stored by user ID and IP address, and provided to anyone requesting them. If credential "privacy" is a requirement, one could secure this connection using IPsec or SSL. To avoid potential deadlocks, the policy daemon is not subject to the connection-filtering mechanism.

6.2 Transparent Firewall

Network bridges are simple devices that transparently connect two or more LAN segments by storing a frame received from one segment and forwarding it to the other segments. More intelligent bridges make use of a spanning tree algorithm to detect and avoid loops in the topology. We have used the basic form of an ethernet bridge in OpenBSD[2], that also provides an IP filtering capability. Thus, the bridge can be used to provide a LAN-transparent firewall between hosts such that no configuration changes are needed on client machines, and only minor changes in network topology are necessary.

For this, we make use of the standard packet filtering mechanism available. As ethernet frames

[2]The bridge code was written by Jason Wright (jason@thought.net), who provided thoughtful comments and considerable testing of all the code described in this section.

pass through the bridge, they are examined to see if they carry IP traffic. If not, the frame is just bridged. If the frame does contain IP traffic, the ethernet header is removed from the frame and copied. The resulting IP packet is passed on to the packet filtering module, which notifies the bridge whether the packet is to be forwarded or dropped. The ethernet header of the frame under examination is appropriately modified on the frame to be forwarded, and the resulting frame is then bridged as normal.

This functionality, useful on its own, can be coupled with IPsec [KA98c], to allow creation of virtual LANs. This is achieved by overlaying an IPsec-protected virtual network on the wide area network (or even the Internet itself). The changes necessary to the bridge and IPsec code for this were fairly minimal, due to compatibility of some design decisions made independently in the development of the two packages.

The enhanced bridge can also be used to provide transparent IPsec gateway capability for a host or even a network. In this mode, the bridge examines transient IP traffic and may, depending on security policy, establish IPsec security associations (SAs) with a remote host pretending to be the local communication endpoint for an IP session[3]. There are two main benefits from this. First, this allows protection of the communications of a host or network without changes to the protected hosts (which may not even be possible, for old, unsupported, or extremely lightweight systems). Second, the security gateway can act as a security policy enforcer, ensuring that incoming and outgoing packets are adequately protected, based on system or network policy.

[3]The term “IP session” is used here loosely to imply a packet flow between two hosts, one of which is on one of the local segments and is “protected” or “supervised”.

6.2.1 Bridging and IPsec

The filtering capabilities offered by the bridge allow its use as a transparent packet filtering firewall. As was the case with traditional firewalls however, filtering by itself is not sufficient in fulfilling network security concerns. Network layer encryption, typically in the form of IPsec, is increasingly used in protecting traffic between networks, hosts, and users. Thus, we decided to augment the filtering bridge with IPsec capabilities.

The first of these configurations, "virtual LAN," is used to transparently and securely connect ethernet segments over a wide area network. This is achieved by encapsulating ethernet frames inside IPsec packets which are then transmitted to a remote bridge that removes the protection and forwards the frames to the local LAN.

The second configuration is what the standards call a "bump in the wire" (BITW) implementation [KA98c], wherein a security gateway (the bridge) transparently implements IPsec on behalf of one or more "protected" hosts. This allows gradual introduction of IPsec in a network without changing the end host configuration or software. This configuration is also a common design feature of network security systems used by the military.

Perhaps more importantly, such a transparent IPsec gateway can be used to enforce security properties for communications between the protected (or supervised) hosts and the rest of the world. Packets traversing the gateway can be examined and, depending on system policy:

- They may be forwarded or dropped, similar to a packet filtering firewall.

- Outgoing packets may cause the security gateway to attempt to establish a security association (SA) with the destination host, pretending to be the originating host, if the packets are un-encrypted. If the packets are already IPsec-protected, it could simply forward them (or, in our case, bridge them). Naturally, the security gateway may always opt to establish an SA with the destination, regardless of the existing security properties of the packet stream.

- Similarly, for incoming packets, the gateway could establish a security association with the originator if the packet was received un-encrypted and/or unauthenticated, again pretending to be the destination host.
- Finally, the bridge can intercept incoming IKE [HC98] packets that request negotiation with one of the protected hosts, and perform the negotiation as that host.

When the SAs use the IP address of the protected host, the bridge is totally transparent to both the protected host and the destination host or gateway. There are two issues that need to be addressed in this configuration however:

- The IPsec standard specifies that IP fragments should not be IPsec-processed in transport mode. That is, fragmentation must occur after IPsec processing has taken place, or tunnel mode (IP-in-IP encapsulation) must be used. Thus, the bridge must either use only tunnel mode IPsec, or reassemble all fragments received from the protected host, IPsec-process the re-constructed packet, then fragment the resulting packet. For performance and simplicity reasons, we decided to use the former approach. The disadvantage is that all IPsec-processed packets are 20 bytes larger (the size of the external IP header).
- Since the remote host is not aware of the encrypting bridge's existence, IPsec packets will be addressed to the protected host or network. Thus, we have to modify the bridge to recognize these addresses and process those IPsec packets. In fact, address recognition is unnecessary. The bridge only has to watch for IPsec packets (transport protocol ESP or AH), and use the information in the packet to perform an SA lookup. If an SA is found, the packet is

processed. Otherwise, it may be dropped or allowed through depending on the filtering configuration.

Left to its own devices, key management will establish an SA using the IP address of the bridge (and thus end up being functionally equivalent to an encrypting gateway). To really hide the bridge from the remote host, the source address of the protected host must be used. Thus the key management daemon, *isakmpd,* has to operate in the "bridge mode," wherein it asks the kernel to use a non-local IP address. This requires a minor change in the *bind()* system call code, to allow for socket binding to non-local addresses. To capture the responses, all UDP traffic to port 500 (the port used by the IKE protocol) is diverted to the bridge *isakmpd.* This is also necessary when remote hosts attempt to initiate an IKE exchange with a protected host. In both cases, *isakmpd* must be informed of and use the "local" address associated with the incoming packet. *isakmpd* also needs the "local" address so as to select the appropriate authentication information. The changes to this effect are fairly minimal.

The combination of packet filtering and SA-on-demand can be used effectively to enforce network security policy for the protected host(s). One particularly interesting configuration involves the bridge establishing SAs for un-encrypted-only traffic; if end-hosts use IPsec or SSL for end-to-end packet security, the bridge simply forwards the packets. In another configuration, the bridge permits all packets through, but attempts to establish SAs for such communications and uses them if the remote hosts can do IPsec (also known as "opportunistic encryption").

Thus, a transparent firewall can be used as a first step in deploying a distributed firewall infrastructure; initially transparent firewalls are deployed protecting subnets or even individual hosts and servers. These act on the behalf of the protected systems, potentially performing application-layer processing (*e.g.,* using a transparent web proxy), both controlling access to the protected systems and using the appropriate security mechanisms to protect their network traffic, without the protected systems' knowledge.

6.3 Future Work

From our discussion in Chapters 4 and 5, several directions for further research have emerged:

- Acquire traffic traces from a deployed system with real users, and correlate these to the security policies that the network needs to enforce. This will allow us to characterize the performance improvement and overall system behavior of our architecture over traditional access control mechanisms. More generally, such measurements would lead to a better understanding of network behavior and the requirements for security mechanisms.
- Further research in high-level policy languages and specification models can lead to better administration tools and methodologies. Presenting specialized management views of the infrastructure suited to particular applications, as discussed in [Pre96], is a promising approach.
- As we discussed in Section 5.3, our policy coordination mechanism (the composition hooks) require an agreement, at the administrator level, of what the semantics of the various values emitted by one policy are. In federated environments, such agreements will have to be established dynamically between administrators that have not previous communicated.
- The policies for applications we have looked at, IPsec and TLS (and other similar protocols, such as SSH) follow the same fundamental access control model: access is denied unless explicitly permitted. It would be interesting to examine how our system can be applied to other types of environments and access control policies.

- Since STRONGMAN allows considerable flexibility in the way policies are distributed and enforced, we can examine models of authorization other than purely hierarchical. Of particular interest are peer-to-peer architectures (where the distinction between users and administrators disappears) and self-organizing networks (*e.g.,* wireless ad-hoc networks, disaster recovery systems, *etc.*).

Chapter 7

Conclusions

Over the past several years, security has become a greater concern for network architects and users. As a result, a variety of security mechanisms have been developed and deployed, in an attempt to provide very fine-grained access control capabilities to administrators. However, such mechanisms are becoming a bottleneck in terms of performance and administrative effort, because they do not take advantage of the benefits of replication and separation of duty respectively. The major contributions of this dissertation are:

- Development of a scalable architecture for access control. This architecture is application-, front-end-, and protocol-independent, and removes the architectural overhead imposed by traditional access control mechanisms, while providing increased flexibility to the administrators, through use of the *trust management model* of authorization.

- The *first* system that demonstrates **improved** performance of the access control functions as a result of decentralization, compared to traditional access control enforcement mechanisms (such as firewalls).

- Development of a prototype implementation in OpenBSD, demonstrating the feasibility of the architecture.

- Performance bottlenecks in this architecture have been identified, and the various tradeoffs involved in credential generation, distribution, and enforcement have been studied both qualitatively and quantitatively.

The following sections will expand upon these contributions.

7.1 Scalable Architecture

The STRONGMAN architecture, discussed in Chapter 4, is the first design that removes the access control enforcement mechanism as the performance and architectural bottleneck in distributed systems. It does so by using the trust-management authorization model, policy compliance checker local to the enforcement points, a simple model for providing policy composition, and the concept of "lazy binding" of policies. As shown in Chapter 4, this approach allows a system to:

1. Improve performance and resource utilization through service replication. Traditional access control systems require replication of the policy to all service instances, or "hardwiring" assignments of users/clients to service instances. By expressing access control policy rules purely in terms of credentials and making sure these rules are presented only when and where needed, we eliminated this bottleneck.

2. Retain flexibility in expressing cross-service policy constraints (*e.g.,* web access decisions based on network layer security mechanism characteristics), without introducing an architectural bottleneck. In our "composition hook" concept, policy dependencies assume a client-server structure where minimal arrangements need to be made in advance (*i.e.,* agreement

on the semantics of specific "hooks").

3. Use arbitrary administrative structures, implementing different separation of duty (SoD) strategies. Separation of duty is the only management approach known to scale, and the trust-management approach allows us to naturally implement any variation of SoD. This is made possible through the natural way KeyNote, our trust-management system, supports delegation.

7.2 Prototype Implementation

A third significant aspect of this work is that parts of the architecture have been embodied as part of the key management daemon in IPsec (`isakmpd`), and of a popular web browser (Apache). The implementation is being widely distributed with the OpenBSD operating system, and is being used in commercial products.

This implementation demonstrates the feasibility (and simplicity) of the STRONGMAN architecture and the ease of converting existing applications and protocols to work as part of a STRONGMAN-enabled network. The implementation is used to measure the performance characteristics of our architecture. The results are summarized in Section 7.3.

7.3 Performance Considerations

The prototype implementation was used to experimentally verify the performance analysis presented in Chapter 5. The analysis shows that:

1. The cost of using KeyNote as part of the policy enforcement mechanism is low: 130 microseconds plus cryptographic algorithm overhead (order of 7 milliseconds, in the test system). The cryptographic costs have to be incurred in any case, if a security protocol is used to protect traffic. This is a one-time cost, incurred when a user first contacts a service: in certain applications (*e.g.,* IPsec), the result of the policy evaluation can be cached and used thereafter at the same speed as the underlying implementation allows; and in any case, the cost of the cryptographic operations is only incurred once.

   Furthermore, given the considerable locality of communication shown in [Mog92, LS90, CDJM91, Ioa93], this initial cost can be easily amortized over the number of packets and bytes exchanged between the user and the service.

2. In certain configurations, end-to-end performance can improve by removing intermediate access control enforcement points (*e.g.,* a firewall) and pushing policy enforcement at the hosts and other end-points. This is the approach taken in the Distributed Firewall work, presented in Chapter 6. Our analysis shows that performance can improve as a result of the reduced processing requirements at each of the individual end-points (which only have to enforce policy on the traffic they see) and by the replacement of general-processing systems (as firewalls typically are) with high-speed networking components (a router, or even bare wire). Such a configuration is possible only by assuming a STRONGMAN-like architecture.

3. The principle of "lazy policy instantiation" allows for much lower resource utilization (in terms of storage and processing) in the enforcement nodes. As shown in Section 5.5.5, systems using this approach can scale more gracefully with the number of policies and users than traditional systems.

7.4 Closing Remarks

This work has a significant impact on the administration and manageability of networks. In this dissertation, I have demonstrated the need and provided an architecture for decentralized access control that allows the administrator to scale system performance using traditional approaches (such as service replication) without sacrificing flexibility or security. I have also shown how this architecture can improve performance in certain commonly-used configurations, with no loss of security (or, indeed, with increased security). Finally, I have analyzed the various tradeoffs with respect to policy generation, distribution, revocation, and enforcement, which allow the customization of the architecture to the particular requirements of a given network. Thus, designers can build large networks which are manageable and whose access control mechanism performance characteristics follow those of the rest of the system, regardless of the network size.

Appendix A

KeyNote Syntax and Semantics

KeyNote is a simple and flexible trust-management system designed to work well for a variety of large- and small-scale Internet-based applications. It provides a single, unified language for both local policies and credentials. KeyNote policies and credentials, called 'assertions', contain predicates that describe the trusted actions permitted by the holders of specific public keys. KeyNote assertions are essentially small, highly-structured programs. A signed assertion, which can be sent over an untrusted network, is also called a 'credential assertion'. Credential assertions, which also serve the role of certificates, have the same syntax as policy assertions but are also signed by the principal delegating the trust.

In KeyNote:

- Actions are specified as a collection of name-value pairs.
- Principal names can be any convenient string and can directly represent cryptographic public keys.
- The same language is used for both policies and credentials.

- The policy and credential language is concise, highly expressive, human readable and writable, and compatible with a variety of storage and transmission media, including electronic mail.

- The compliance checker returns an application-configured ‘policy compliance value’ that describes how a request should be handled by the application. Policy compliance values are always positively derived from policy and credentials, facilitating analysis of KeyNote-based systems.

- Compliance checking is efficient enough for high-performance and real-time applications.

We assume that applications communicate with a locally trusted KeyNote compliance checker via a ‘function call’ style interface, sending a collection of KeyNote policy and credential assertions plus an action description as input and accepting the resulting policy compliance value as output. However, the requirements of different applications, hosts, and environments may give rise to a variety of different interfaces to KeyNote compliance checkers.

KeyNote Concepts

In KeyNote, the authority to perform trusted actions is associated with one or more ‘principals’. A principal may be a physical entity, a process in an operating system, a public key, or any other convenient abstraction. KeyNote principals are identified by a string called a ‘Principal Identifier’. In some cases, a Principal Identifier will contain a cryptographic key interpreted by the KeyNote system (*e.g.,* for credential signature verification). In other cases, Principal Identifiers may have a structure that is opaque to KeyNote.

Principals perform two functions of concern to KeyNote: they request ‘actions’ and they issue ‘assertions’. Actions are any trusted operations that an application places under KeyNote control. Assertions delegate the authorization to perform actions to other principals.

Actions are described to the KeyNote compliance checker in terms of a collection of name-value pairs called an ‘action attribute set’. The action attribute set is created by the invoking application. Its structure and format are described in detail in the section on Action Attributes later in this Appendix.

KeyNote provides advice to applications about the interpretation of policy with regard to specific requested actions. Applications invoke the KeyNote compliance checker by issuing a ‘query’ containing a proposed action attribute set and identifying the principal(s) requesting it. The KeyNote system determines and returns an appropriate ‘policy compliance value’ from an ordered set of possible responses.

The policy compliance value returned from a KeyNote query advises the application how to process the requested action. In the simplest case, the compliance value is Boolean (*e.g.,* “reject” or “approve”). Assertions can also be written to select from a range of possible compliance values, when appropriate for the application (*e.g.,* “no access”, “restricted access”, “full access”). Applications can configure the relative ordering (from ‘weakest’ to ‘strongest’) of compliance values at query time.

Assertions are the basic programming unit for specifying policy and delegating authority. Assertions describe the conditions under which a principal authorizes actions requested by other principals. An assertion identifies the principal that made it, which other principals are being authorized, and the conditions under which the authorization applies. The syntax of assertions can be found in [BFIK99c].

A special principal, whose identifier is “POLICY”, provides the root of trust in KeyNote.

"POLICY" is therefore considered to be authorized to perform any action. Assertions issued by the "POLICY" principal are called 'policy assertions' and are used to delegate authority to otherwise untrusted principals. The KeyNote security policy of an application consists of a collection of policy assertions.

When a principal is identified by a public key, it can digitally sign assertions and distribute them over untrusted networks for use by other KeyNote compliance checkers. These signed assertions are also called 'credentials', and serve a role similar to that of traditional public key certificates. Policies and credentials share the same syntax and are evaluated according to the same semantics. A principal can therefore convert its policy assertions into credentials simply by digitally signing them.

KeyNote is designed to encourage the creation of human-readable policies and credentials that are amenable to transmission and storage over a variety of media. Its assertion syntax is inspired by the format of RFC822-style message headers [Cro82]. A KeyNote assertion contains a sequence of sections, called 'fields', each of which specifies one aspect of the assertion's semantics. Fields start with an identifier at the beginning of a line and continue until the next field is encountered. For example:

```
KeyNote-Version: 2
Comment: A simple, if contrived, email certificate for user mab
Local-Constants:  ATT_CA_key = ``rsa-hex:acdfa1df1011bbac''
                  mab_key = ``dsa-hex:deadbeefcafe001a''
Authorizer: ATT_CA_key
Licensees: mab_key
Conditions: ((app_domain == ``email'')  # valid for email only
            && (address == ``mab@research.att.com''));
Signature: ``sig-rsa-sha1-hex:f00f2244''
```

With respect to syntax, the important fields in an assertion are:

- *Authorizer* contains the public key of the issuer of the credential. The public key can be used to verify the signature encoded in the *Signature* field. Policy assertions are those with the "POLICY" keyword in the Authorizer field.
- *Licensees* contains an expression of public keys, to which some privilege is granted. In the simple case, it contains one public key; more generally, it contains a list of keys that must be authorize a request.
- The *Conditions* field contains one or more boolean expressions that authorize (or not) a particular request based on the details of that request, by examining the action attributes (as we shall see the section on Action Attributes.

For more details on the syntax, see [BFIK99c].

KeyNote semantics resolve the relationship between an application's policy and actions requested by other principals, as supported by credentials. The KeyNote compliance checker processes the assertions against the action attribute set to determine the policy compliance value of a requested action. These semantics are defined in a later section.

An important principle in the design of KeyNote is *assertion monotonicity;* the policy compliance value of an action is always positively derived from assertions made by trusted principals. Removing an assertion never results in increasing the compliance value returned by KeyNote for a given query. The monotonicity property can simplify the design and analysis of complex network-based security protocols; network failures that prevent the transmission of credentials can never result in spurious authorization of dangerous actions. A detailed discussion of monotonicity and safety in trust management can be found in [BFL96] and [BFS98].

Action Attributes

Actions to be evaluated by KeyNote are described by a collection of name-value pairs called the *action attribute set*. Action attributes are the mechanism by which applications communicate requests to KeyNote and are the primary objects on which KeyNote assertions operate. An action attribute set is passed to the KeyNote compliance checker with each query.

Each action attribute consists of a name and a value. The semantics of the names and values are not interpreted by KeyNote itself; they vary from application to application and must be agreed upon by the writers of applications and the writers of the policies and credentials that will be used by them.

Action attribute names and values are represented by arbitrary-length strings. KeyNote guarantees support of attribute names and values up to 2048 characters long. The handling of longer attribute names or values is not specified and is KeyNote-implementation-dependent. Applications and assertions should therefore avoid depending on the the use of attributes with names or values longer than 2048 characters. The length of an attribute value is represented by an implementation-specific mechanism (*e.g.,* null-terminated strings, an explicit length field, *etc.*).

Attribute values are inherently untyped and are represented as character strings by default. Attribute values may contain any non-null ASCII character. Numeric attribute values should first be converted to an ASCII text representation by the invoking application, *e.g.,* the value 1234.5 would be represented by the string “1234.5”.

Attribute names are of the form:

```
<AttributeID>:: {Any string starting with a-z, A-Z, or the
underscore character, followed by any number of
a-z, A-Z, 0-9, or underscore characters} ;
```

That is, an *AttributeID* begins with an alphabetic or underscore character and can be followed by any number of alphanumerics and underscores. Attribute names are case sensitive. The exact mechanism for passing the action attribute set to the compliance checker is determined by the KeyNote implementation. Depending on specific requirements, an implementation may provide a mechanism for including the entire attribute set as an explicit parameter of the query, or it may provide some form of callback mechanism invoked as each attribute is dereferenced, *e.g.,* for access to kernel variables. If an action attribute is not defined its value is considered to be the empty string.

Attribute names beginning with the "_" character are reserved for use by the KeyNote runtime environment and cannot be passed from applications as part of queries. [BFIK99c] defines some of these special names.

The names of other attributes in the action attribute set are not specified by KeyNote but must be agreed upon by the writers of any policies and credentials that are to inter-operate in a specific KeyNote query evaluation.

By convention, the name of the application domain over which action attributes should be interpreted is given in the attribute named *app_domain.* The IANA (or some other suitable authority) will provide a registry of reserved *app_domain* names. The registry will list the names and meanings of each application's attributes.

The *app_domain* convention helps to ensure that credentials are interpreted as they were intended. An attribute with any given name may be used in many different application domains but might have different meanings in each of them. However, the use of a global registry is not always required for small-scale, closed applications; the only requirement is that the policies and credentials made available to the KeyNote compliance checker interpret attributes according to the same semantics assumed by the application that created them.

For example, an email application may reserve the *app_domain* “RFC822-EMAIL” and may use the attributes named *address* (the email address of a message’s sender), *name* (the human name of the message sender), and any *organization* headers present (the organization name). The values of these attributes would be derived in the obvious way from the email message headers. The public key of the message’s signer would be given in the “_ACTION_AUTHORIZERS” attribute.

Note that “RFC822-EMAIL” is a hypothetical example; such a name may or may not appear in the actual registry with these or different attributes.

Query Evaluation Semantics

The KeyNote compliance checker finds and returns the Policy Compliance Value of queries, as defined in the Policy Compliance section below.

Query Parameters

A KeyNote query has four parameters:

- The identifier of the principal(s) requesting the action.
- The action attribute set describing the action.
- The set of compliance values of interest to the application, ordered from _MIN_TRUST to _MAX_TRUST

- The policy and credential assertions that should be included in the evaluation.

The mechanism for passing these parameters to the KeyNote evaluator is application dependent. In particular, an evaluator might provide for some parameters to be passed explicitly, while others are looked up externally (*e.g.,* credentials might be looked up in a network- based distribution system), while still others might be requested from the application as needed by the evaluator, through a 'callback' mechanism (*e.g.,* for attribute values that represent values from among a very large namespace).

Action Requester

At least one Principal must be identified in each query as the 'requester' of the action. Actions may be requested by several principals, each considered to have individually requested it. This allows policies that require multiple authorizations, *e.g.,* 'two person control'. The set of authorizing principals is made available in the special attribute "_ACTION_AUTHORIZERS"; if several principals are authorizers, their identifiers are separated with commas.

Ordered Compliance Value Set

The set of compliance values of interest to an application (and their relative ranking to one another) is determined by the invoking application and passed to the KeyNote evaluator as a parameter of the query. In many applications, this will be Boolean, *e.g.,* the ordered sets FALSE, TRUE or REJECT, APPROVE. Other applications may require a range of possible values, *e.g.,* No_Access, Limited_Access, Full_Access. Note that applications should include in this set only compliance value names that are actually returned by the assertions.

The lowest-order and highest-order compliance value strings given in the query are available in the special attributes named "_MIN_TRUST" and "_MAX_TRUST", respectively. The complete set of query compliance values is made available in ascending order (from _MIN_TRUST to _MAX_TRUST) in the special attribute named "_VALUES". Values are separated with commas; applications that use assertions that make use of the _VALUES attribute should therefore avoid the use of compliance value strings that themselves contain commas.

Principal Identifier Normalization

Principal identifier comparisons among Cryptographic Principal Identifiers (that represent keys) in the Authorizer and Licensees fields or in an action's direct authorizers are performed after normalizing them by conversion to a canonical form.

Every cryptographic algorithm used in KeyNote defines a method for converting keys to their canonical form and that specifies how the comparison for equality of two keys is performed. If the algorithm named in the identifier is unknown to KeyNote, the identifier is treated as opaque. Opaque identifiers are compared as case-sensitive strings.

Notice that use of opaque identifiers in the Authorizer field requires that the assertion's integrity be locally trusted (since it cannot be cryptographically verified by the compliance checker).

Policy Compliance Value Calculation

The Policy Compliance Value of a query is the Principal Compliance Value of the principal named "POLICY". This value is defined as follows:

Principal Compliance Value

The Compliance Value of a principal $< X >$ is the highest order (maximum) of:

- the Direct Authorization Value of principal $< X >$; and
- the Assertion Compliance Values of all assertions identifying $< X >$ in the Authorizer field.

Direct Authorization Value

The Direct Authorization Value of a principal $< X >$ is _MAX_TRUST if $< X >$ is listed in the query as an authorizer of the action. Otherwise, the Direct Authorization Value of $< X >$ is _MIN_TRUST.

Assertion Compliance Value

The Assertion Compliance Value of an assertion is the lowest order (minimum) of the assertion's Conditions Compliance Value and its Licensee Compliance Value.

Conditions Compliance Value

The Conditions Compliance Value of an assertion is the highest-order (maximum) value among all successful clauses listed in the conditions section.

If no clause's test succeeds or the Conditions field is empty, an assertion's Conditions Compliance Value is considered to be the _MIN_TRUST value, as defined Section 5.1. If an assertion's Conditions field is missing entirely, its Conditions Compliance Value is considered to be the _MAX_TRUST value, as defined in Section 5.1.

The set of successful test clause values is calculated as follows:

Recall from the grammar of section 4.6.5 that each clause in the conditions section has two logical parts: a 'test' and an optional 'value', which, if present, is separated from the test with the "$\rightarrow$" token. The test subclause is a predicate that either succeeds (evaluates to logical 'true') or fails (evaluates to logical 'false'). The value subclause is a string expression that evaluates to one value from the ordered set of compliance values given with the query. If the value subclause is missing, it is considered to be _MAX_TRUST. That is, the clause

```
    foo == ''bar'';
```

is equivalent to

```
    foo == ''bar'' -> _MAX_TRUST;
```

If the value component of a clause is present, in the simplest case it contains a string expression representing a possible compliance value. For example, consider an assertion with the following Conditions field:

```
  Conditions:
     @user_id == 0 -> ``full_access'';          # clause (1)
     @user_id < 1000 -> ``user_access'';        # clause (2)
     @user_id < 10000 -> guest_access'';        # clause (3)
     user_name == ``root'' -> ``full_access''; # clause (4)
```

Here, if the value of the "user_id" attribute is "1073" and the "user_name" attribute is "root", the possible compliance value set would contain the values "guest_access" (by clause (3)) and "full_access" (by clause (4)). If the ordered set of compliance values given in the query (in ascending order) is "no_access", "guest_access", "user_access", "full_access", the Conditions Compliance Value of the assertion would be "full_access" (because "full_access" has a higher-order value than "guest_access"). If the "user_id" attribute had the value "19283" and the "user_name" attribute had the value "nobody", no clause would succeed and the Conditions Compliance Value would be "no_access", which is the lowest-order possible value (_MIN_TRUST).

If a clause lists an explicit value, its value string must be named in the query ordered compliance value set. Values not named in the query compliance value set are considered equivalent to _MIN_TRUST.

The value component of a clause can also contain recursively-nested clauses. Recursively-nested clauses are evaluated only if their parent test is true. That is,

```
a == ''b'' ->  { b == ''c'' -> ``value1'';
                 d == ''e''  -> ``value2'';
                 true -> ``value3''; } ;
```

is equivalent to

```
(a == ''b'') && (b == ''c'') -> ``value1'';
(a == ''b'') && (d == ''e'') -> ``value2'';
(a == ''b'') -> ``value3'';
```

String comparisons are case-sensitive.

A regular expression comparison ("~=") is considered true if the left-hand-side string expression matches the right-hand-side regular expression. If the POSIX regular expression group matching scheme is used, the number of groups matched is placed in the temporary meta- attribute "_0" (dereferenced as _0), and each match is placed in sequence in the temporary attributes (_1, _2, ..., _N). These match-attributes' values are valid only within subsequent references made within the same clause. Regular expression evaluation is case- sensitive.

A runtime error occurring in the evaluation of a test, such as division by zero or an invalid regular expression, causes the test to be considered false. For example:

```
foo == ``bar'' -> {
                  @a == 1/0 -> ``oneval'';     # subclause 1
                  @a == 2 -> ``anotherval'';  # subclause 2
                };
```

Here, subclause 1 triggers a runtime error. Subclause 1 is therefore false (and has the value _MIN_TRUST). Subclause 2, however, would be evaluated normally.

An invalid $< RegExpr >$ is considered a runtime error and causes the test in which it occurs to be considered false.

Licensee Compliance Value

The Licensee Compliance Value of an assertion is calculated by evaluating the expression in the Licensees field, based on the Principal Compliance Value of the principals named there.

If an assertion's Licensees field is empty, its Licensee Compliance Value is considered to be _MIN_TRUST. If an assertion's Licensees field is missing altogether, its Licensee Compliance Value is considered to be _MAX_TRUST.

For each principal named in the Licensees field, its Principal Compliance Value is substituted for its name. If no Principal Compliance Value can be found for some named principal, its name is substituted with the _MIN_TRUST value.

The licensees expression (as defined in Section 4.6.4) is evaluated as follows:

- A "(...)" expression has the value of the enclosed subexpression.

- A “&&” expression has the lower-order (minimum) of its two subexpression values.
- A “$||''$ expression has the higher-order (maximum) of its two subexpression values.
- A “ $< K > -of(< List >)''$ expression has the K-th highest order compliance value listed in $< list >$. Values that appear multiple times are counted with multiplicity. For example, if K = 3 and the orders of the listed compliance values are (0, 1, 2, 2, 3), the value of the expression is the compliance value of order 2.

For example, consider the following Licensees field:

```
Licensees: (``alice'' && ``bob'') || ``eve''
```

If the Principal Compliance Value is “yes” for principal “alice”, “no” for principal “bob”, and “no” for principal “eve”, and “yes” is higher order than “no” in the query’s Compliance Value Set, then the resulting Licensee Compliance Value is “no”. Observe that if there are exactly two possible compliance values (*e.g.,* “false” and “true”), the rules of Licensee Compliance Value resolution reduce exactly to standard Boolean logic.

Assertion Management

Assertions may be either signed or unsigned. Only signed assertions should be used as credentials or transmitted or stored on untrusted media. Unsigned assertions should be used only to specify policy and for assertions whose integrity has already been verified as conforming to local policy by some mechanism external to the KeyNote system itself (*e.g.,* X.509 certificates converted to KeyNote assertions by a trusted conversion program).

Implementations that permit signed credentials to be verified by the KeyNote compliance checker generally provide two ‘channels’ through which applications can make assertions available. Unsigned, locally-trusted assertions are provided over a ‘trusted’ interface, while signed credentials are provided over an ‘untrusted’ interface. The KeyNote compliance checker verifies correct signatures for all assertions submitted over the untrusted interface. The integrity of KeyNote evaluation requires that only assertions trusted as reflecting local policy are submitted to KeyNote via the trusted interface.

Note that applications that use KeyNote exclusively as a local policy specification mechanism need use only trusted assertions. Other applications might need only a small number of infrequently changed trusted assertions to ‘bootstrap’ a policy whose details are specified in signed credentials issued by others and submitted over the untrusted interface.

Implementation Issues

Informally, the semantics of KeyNote evaluation can be thought of as involving the construction a directed graph of KeyNote assertions rooted at a POLICY assertion that connects with at least one of the principals that requested the action.

Delegation of some authorization from principal $< A >$ to a set of principals $< B >$ is expressed as an assertion with principal $< A >$ given in the Authorizer field, principal set $< B >$ given in the Licensees field, and the authorization to be delegated encoded in the Conditions field. How the expression digraph is constructed is implementation-dependent and implementations may use different algorithms for optimizing the graph’s construction. Some implementations might use a ‘bottom up’ traversal starting at the principals that requested the action, others might follow a ‘top down’ approach starting at the POLICY assertions, and still others might employ other heuristics entirely.

Implementations are encouraged to employ mechanisms for recording exceptions (such as division by zero or syntax error), and reporting them to the invoking application if requested. Such mechanisms are outside the scope of this paper.

Appendix B

IPsec

While IP has proven to be an efficient and robust protocol when it comes to actually getting data across the Internet, it does not inherently provide any protection of that data. There are no facilities to provide confidentiality, or to ensure the integrity or authenticity of IP [Pos81] datagrams. In order to remedy the security weaknesses of IP, a pair of protocols collectively called IP Security, or IPsec [Atk95c, KA98c] for short, has been standardized by the IETF. The protocols are ESP (Encapsulating Security Payload) [Atk95b, KA98b] and AH (Authentication Header) [Atk95a, KA98a]. Both provide integrity, authenticity, and replay protection, while ESP adds confidentiality to the picture. IPsec can also be made to protect IP datagrams for traffic that does not originate or terminate to the host doing the IPsec processing. The IPsec endpoints in this arrangement thereby become security gateways and take part in a virtual private network (VPN) where ordinary IP packets are tunneled inside IPsec [Sim95].

Network-layer security has a number of very important advantages over security at other layers of the protocol stack. Network-layer protocols are generally hidden from applications, which can therefore automatically take advantage of whatever network-layer encryption services that host provides. Most importantly, network-layer protocols offer a remarkable flexibility not available at higher or lower layers. They can provide security on an end-to-end basis (securing the data between two hosts), route-to-route (securing data passing over a particular set of links), edge-to-edge (securing data as it passes from a “secure” network to an “insecure” one), or a combination of these.

IPsec is based on the concept of *datagram encapsulation.* Cryptographically protected network layer packets are placed inside (as the payload of) other network packets, making the encryption transparent to any intermediate nodes that must process packet headers for routing, *etc.* Outgoing packets are encapsulated, encrypted, and authenticated (as appropriate) just before being sent to the network, and incoming packets are verified, decrypted, and decapsulated immediately upon receipt[IB93]. Key management in such a protocol is straightforward in the simplest case. Two hosts can use any key-agreement protocol to negotiate keys with one another, and use those keys as part of the encapsulating and decapsulating packet transforms.

IPsec “connections” are described in a data structure called a *security association (SA)*. Encryption and authentication keys are contained in the SA at each endpoint, and each IPsec-protected packet has an SA identifier that indexes the SA database of its destination host (note that not all SAs specify both encryption and authentication; authentication-only SAs are commonly used, and encryption-only SAs are possible, albeit considered insecure).

Wire Protocols

IPsec packets include an extra header between the IP header and the payload. This header holds IPsec-specific data, such as an anti-replay sequence number, cryptographic synchronization data, and integrity check values. If the security protocol in use is ESP, a cryptographic transform is applied to the payload in-place, effectively hiding the data. As an example, an UDP datagram

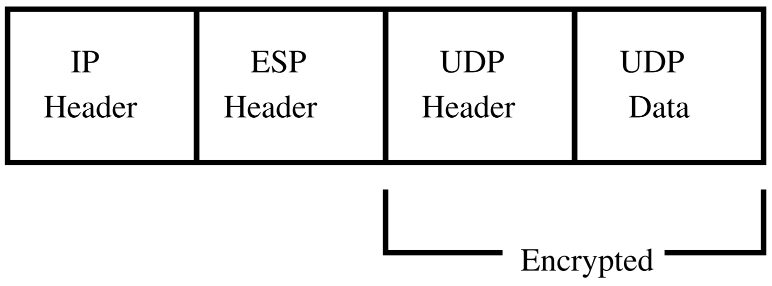


Figure 9.1: IPsec Transport Mode.

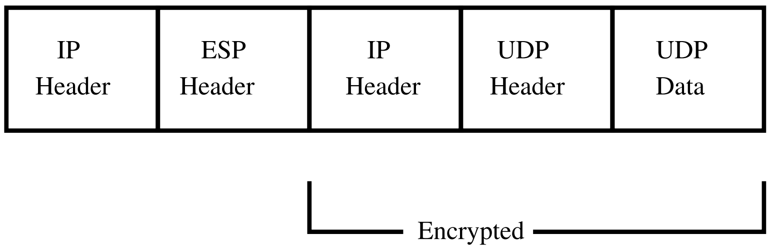


Figure 9.2: IPsec Tunnel Mode.

protected by ESP is shown in figure 9.1.

This mode of operation is called transport mode, as opposed to tunnel mode. Tunnel mode is typically used when a security gateway is protecting datagrams for other hosts. Tunnel mode differs from transport mode by the addition of a new, outer, IP header consisting of the security gateways' addresses instead of the actual source and destination, as shown in figure 9.2.

Key Management

IPsec provides a solution to the problem of securing communications. However, for large-scale deployment and use, an automated method for managing SAs and key setup is required. There are several issues in this problem domain, such as negotiation of SA attributes, authentication, secure key distribution, and key aging. Manual management is complicated, tedious, error-prone, and does not scale. Standardized protocols addressing these issues are needed; IETF's recommended protocol is named IKE[HC98], the Internet Key Exchange. IKE is based on a framework protocol called ISAKMP[MSST98] and implements semantics from the Oakley key exchange, therefore IKE is also known as ISAKMP/Oakley.

The IKE protocol has two phases: the first phase establishes a secure channel between the two key management daemons, while in the second phase IPsec SAs can be directly negotiated. The first phase negotiates at least an authentication method, an encryption algorithm, a hash algorithm, and a Diffie-Hellman group. This set of parameters is called a "Phase 1 SA." Using this information, the peers authenticate each other and compute key material to use for protecting Phase 2. Depending on the protection suite specified during Phase 1, different modes can be used to establish a Phase 1 SA, the two most important ones being "main mode" and "aggressive mode." Main mode provides identity protection, by encrypting the identities of the peers before transmission. Aggressive mode provides somewhat weaker guarantees, but requires fewer messages and allows for "road warrior" [1] types of configuration using passphrase-based authentication.

The second phase is commonly called "quick mode" and results in a IPsec SA tuple (one incoming and one outgoing). Quick mode is protected by a Phase 1 SA, therefore it does not need to provide its own authentication protection. This allows for a fast negotiation (hence the name).

[1]Remote mobile users that need to access the protected network behind a firewall, using IPsec.

Appendix C

KeyNote Action Attributes for IPsec

All the data in the fields of IKE packets are passed to KeyNote as *action attributes*; these attributes are available to the Conditions sections of the KeyNote assertions. There are a number of attributes defined (the complete list appears in the `isakmpd.policy` man page in OpenBSD 2.6 and later). The most important attributes include:

app_domain is always set to `IPsec policy.`

pfs is set to `yes` if a Diffie-Hellman exchange will be performed during Quick Mode, otherwise it is set to `no`.

ah_present, esp_present, comp_present are set to `yes` if an AH, ESP, or compression proposal was received in IKE (or other key management protocol), and to `no` otherwise. Note that more than one of these may be set to yes, since it is possible for an IKE proposal to specify "SA bundles" (combinations of ESP and AH that must be applied together).

esp_enc_alg is set to one of `des, des-iv64, 3des, rc4, idea` and so on depending on the proposed encryption algorithm to be used in ESP.

local_ike_address, remote_ike_address are set to the IPv4 or IPv6 address (expressed as a dotted-decimal notation with three-digit, zero-prefixed octets (*e.g.,* 010.010.003.045)) of the local interface used in the IKE exchange, and the address of the remote IKE daemon, respectively.

remote_filter, local_filter are set to the IPv4 or IPv6 addresses proposed as the remote and local User Identities in Quick Mode. Host addresses, subnets, or address ranges may be expressed (and thus controlled by policy).

Bibliography

[ABG$^+$98] C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra, and C. Villamizer. Routing Policy Specification Language (RPSL). Request for Comments (Proposed Standard) 2280, Internet Engineering Task Force, January 1998.

[Arb99] W. A. Arbaugh. *Chaining Layered Integrity Checks*. PhD thesis, University of Pennsylvania, Philadelphia, 1999.

[AS99] G.J. Ahn and R. Sandhu. The RSL99 language for role-based separation of duty constraints. In *Proceedings of the 4th ACM Workshop on Role-Based Acess Control (RBAC)*, pages 43–54, October 1999.

[Atk95a] R. Atkinson. IP Authentication Header. RFC 1826, August 1995.

[Atk95b] R. Atkinson. IP Encapsulating Security Payload. RFC 1827, August 1995.

[Atk95c] R. Atkinson. Security Architecture for the Internet Protocol. RFC 1825, August 1995.

[Aur99] T. Aura. Distributed access rights management with delegation certificates. In *Secure Internet Programming* [VJ99], pages 211–235.

[BC99] Paul Barford and Mark Crovella. Measuring web performance in the wide area. Technical Report 1999-004, 23, 1999.

[BCD$^+$00] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) Protocol. Request for comments (proposed standard), Internet Engineering Task Force, January 2000.

[BdVS00] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. A Modular Approach to Composing Access Policies. In *Proceedings of Computer and Communications Security (CCS) 2000*, pages 164–173, November 2000.

[Bel99] S. M. Bellovin. Distributed Firewalls. *;login: magazine, special issue on security*, November 1999.

[BFIK99a] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming* [VJ99], pages 185–210.

[BFIK99b] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The keynote trust management system version 2. Internet RFC 2704, September 1999.

[BFIK99c] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. Internet RFC 2704, September 1999.

[BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, Los Alamitos, 1996.

[BFRS97] M. Blaze, J. Feigenbaum, P. Resnick, and M. Strauss. Managing Trust in an Information Labeling System. In *European Transactions on Telecommunications, 8*, pages 491–501, 1997.

[BFS98] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance Checking in the PolicyMaker Trust-Management System. In *Proc. of the Financial Cryptography '98, Lecture Notes in Computer Science, vol. 1465*, pages 254–274. Springer, Berlin, 1998.

[BGS92] J.A. Bull, L. Gong, and K.R. Sollins. Towards Security in an Open Systems Federation. In *Lecture Note in Computer Science 648, ESORICS '92*, pages 3–20. Springer-Verlag, 1992.

[BIK01a] M. Blaze, J. Ioannidis, and A. D. Keromytis. Offline Micropayments without Trusted Hardware. In *Proceedings of the Fifth International Conference on Financial Cr yptography*, 2001.

[BIK01b] M. Blaze, J. Ioannidis, and A.D. Keromytis. Trust Managent for IPsec. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, pages 139–151, February 2001.

[BKRY99] S. Bhatt, A.V. Konstantinou, S.R. Rajagopalan, and Y. Yemini. Managing Security in Dynamic Networks. In *Proceedings of the 13th USENIX Systems Administration Conference (LISA)*, November 1999.

[BMNW99] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: a novel firewall management toolkit. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 17–31, May 1999.

[CB94] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.

[CC97] L. Cholvy and F. Cuppens. Analyzing consistency of security policies. In *RSP: 18th IEEE Computer Society Symposium on Research in Security and Privacy*, 1997.

[CCI89] CCITT. *X.509: The Directory Authentication Framework*. International Telecommunications Union, Geneva, 1989.

[CDJM91] R. Caceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. Characteristics of wide-area TCP/IP conversations. *ACM SIGCOMM Computer Communication Review*, 21(4):101–112, September 1991.

[CFL$^+$97] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust Management for Web Applications. In *World Wide Web Journal, 2*, pages 127–139, 1997.

[CL98] M. Carney and B. Loe. A Comparison of Methods for Implementing Adaptive Security Policies. In *Proceedings of the 7th USENIX Security Symposium*, January 1998.

[Cla88] D. D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proc. SIGCOMM 1988*, pages 106–114, 1988.

[CLZ99] M. Condell, C. Lynn, and J. Zao. Security Policy Specification Language. Internet draft, Internet Engineering Task Force, July 1999.

[CRAG99] P. Calhoun, A. Rubens, H. Akhtar, and E. Guttman. DIAMETER Base Protocol. Internet Draft, Internet Engineering Task Force, December 1999. Work in progress.

[Cro82] D. H. Crocker. Standard For The Format Of ARPA Internet Text Messages. Request for Comments 822, Internet Engineering Task Force, August 1982.

[CS96] J. Chinitz and S. Sonnenberg. A Transparent Security Framework For TCP/IP and Legacy Applications. Technical report, Intellisoft Corp., August 1996.

[DA97] Tim Dierks and Christopher Allen. The TLS PRotocol Version 1.0. Work In Progress, November 1997.

[DA99] T. Dierks and C. Allen. The TLS protocol version 1.0. Request for Comments (Proposed Standard) 2246, Internet Engineering Task Force, January 1999.

[Dah95] M. Dahlin. *Serverless Network File Systems*. PhD thesis, University of California, Berkeley, December 1995.

[DH96] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. Internet RFC 1883, January 1996.

[DOD85] DOD. Trusted Computer System Evaluation Criteria. Technical Report DOD 5200.28-STD, Department of Defense, December 1985.

[dRHG$^+$99] T. de Raadt, N. Hallqvist, A. Grabowski, A. D. Keromytis, and N. Provos. Cryptography in OpenBSD: An Overview. In *Proc. of the 1999 USENIX Annual Technical Conference, Freenix Track*, pages 93 – 101, June 1999.

[Edi] RFC Editor. RFCs issued by year. http://www.rfceditor.org/num_rfc_year.html.

[EFL$^+$99] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI certificate theory. Request for Comments 2693, Internet Engineering Task Force, September 1999.

[Ell99] C. Ellison. SPKI requirements. Request for Comments 2692, Internet Engineering Task Force, September 1999.

[Eps99] J. Epstein. Architecture and Concepts of the ARGuE Guard. In *Proc. of the 15th Annual Computer Security Applications Conference (ACSAC)*, December 1999.

[GSSC96] M. Greenwald, S.K. Singhal, J.R. Stone, and D.R. Cheriton. Designing an Academic Firewall. Policy, Practice and Experience with SURF. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, pages 79–91, February 1996.

[Gut97] Joshua D. Guttman. Filtering Postures: Local Enforcement for Global Policies. In *IEEE Security and Privacy Conference*, pages 120–129, May 1997.

[HBM98] R.J. Hayton, J.M. Bacon, and K. Moody. Access Control in an Open Distributed Environment. In *IEEE Symposium on Security and Privacy*, May 1998.

[HC98] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force, November 1998.

[HGPS99] J. Hale, P. Galiasso, M. Papa, and S. Shenoi. Security Policy Coordination for Heterogeneous Information Systems. In *Proc. of the 15th Annual Computer Security Applications Conference (ACSAC)*, December 1999.

[Hin99] S. Hinrichs. Policy-Based Management: Bridging the Gap. In *Proc. of the 15th Annual Computer Security Applications Conference (ACSAC)*, December 1999.

[HK00] Niklas Hallqvist and Angelos D. Keromytis. Implementing Internet Key Exchange (IKE). In *Proceedings of the Annual USENIX Technical Conference, Freenix Track*, pages 201–214, June 2000.

[HKM$^+$98] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. PLAN: A Programming Language for Active Networks. Technical Report MS-CIS-98-25, Department of Computer and Information Science, University of Pennsylvania, February 1998.

[How97] John D. Howard. *An Analysis Of Security On The Internet 1989 - 1995*. PhD thesis, Carnegie Mellon University, April 1997.

[IB93] John Ioannidis and Matt Blaze. The Architecture and Implementation of Network-Layer Security Under Unix. In *Fourth Usenix Security Symposium Proceedings*. USENIX, October 1993.

[IKBS00] S. Ioannidis, A.D. Keromytis, S.M. Bellovin, and J.M. Smith. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS) 2000*, pages 190–199, November 2000.

[Ioa93] J. Ioannidis. *Protocols for Mobile Networking*. PhD thesis, Columbia University, New York, 1993.

[Jae99] Trent Jaeger. On the Increasing Importance of Constraints. In *Proceedings of the 4th ACM Workshop on Role-Based Acess Control (RBAC)*, pages 33–42, October 1999.

[KA98a] S. Kent and R. Atkinson. IP Authentication Header. Request for Comments (Proposed Standard) 2402, Internet Engineering Task Force, November 1998.

[KA98b] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). Request for Comments (Proposed Standard) 2406, Internet Engineering Task Force, November 1998.

[KA98c] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Request for Comments (Proposed Standard) 2401, Internet Engineering Task Force, November 1998.

[Kan01] Y. Kanada. Taxonomy and Description of Policy Combination Methods. In *Proceedings of the 2001 International Workshop on Policies for Distributed Systems and Networks*, pages 171–184, January 2001.

[Lab93] RSA Laboratories. *PKCS #1: RSA Encryption Standard*, version 1.5 edition, November 1993.

[Lam71] B.W. Lampson. Protection. In *Proc. of the 5th Princeton Symposium on Information Sciences and Systems*, pages 473–443, March 1971.

[Lam74] B.W. Lampson. Protection. *Operating Systems Review*, 8(1):18–24, January 1974.

[LeF92] W. LeFebvre. Restricting network access to system daemons under SunOS. In *Proceedings of the Third USENIX UNIX Security Symposium*, pages 93–103, 1992.

[Li00] Ninghui Li. *Delegation Logic: A Logic-based Approach to Distributed Authorization*. PhD thesis, 2000.

[LMB] R. Levien, L. McCarthy, and M. Blaze. Transparent Internet E-mail Security. http://www.cs.umass.edu/~lmccarth/crypto/papers/email.ps.

[LS90] M. J. Lorence and M. Satyanarayanan. IPwatch: a tool for monitoring network locality. *ACM SIGOPS Operating Systems Review*, 24(1):58–80, January 1990.

[LSM97] J. Lacy, J. Snyder, and D. Maher. Music on the Internet and the Intellectual Property Protection Problem. In *Proc. of the International Symposium on Industrial Electronics*, pages SS77–83. IEEE Press, 1997.

[MAM$^+$99] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 internet public key infrastructure online certificate status protocol - OCSP. Request for Comments 2560, Internet Engineering Task Force, June 1999.

[MJ93] Steven McCanne and Van Jacobson. A BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of USENIX Winter Technical Conference*, pages 259–269, San Diego, California, January 1993. Usenix.

[MJ00] Patrick McDaniel and Sugih Jamin. Windowed certificate revocation. In *Proceedings of Infocom*, Tel Aviv, Israel, March 2000.

[MNSS87] S. P. Miller, B. C. Neuman, J. I. Schiller, and J. H. Saltzer. Kerberos Authentication and Authorization System. Technical report, MIT, December 1987.

[Mog89] J. C. Mogul. Simple and flexible datagram access controls for UNIX-based gateways. In *Proceedings of the USENIX Summer 1989 Conference*, pages 203–221, 1989.

[Mog92] J. C. Mogul. Network locality at the scale of processes. 10(2):81–109, May 1992.

[Mol95] A. Molitor. An Architecture for Advanced Packet Filtering. In *Proceedings of the 5th USENIX UNIX Security Symposium*, June 1995.

[MRA87] J. Mogul, R. Rashid, and M. Accetta. The Packet Filter: An Efficient Mechanism for User-level Network Code. In *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, pages 39–51, November 1987.

[MSST98] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet security association and key management protocol (ISAKMP). Request for Comments (Proposed Standard) 2408, Internet Engineering Task Force, November 1998.

[MWL95] B. McKenney, D. Woycke, and W. Lazear. A Network of Firewalls: An Implementation Example. In *Proceedings of the 11th Anual Computer Security Applications Conference (ACSAC)*, pages 3–13, December 1995.

[NBS77] Data Encryption Standard, January 1977.

[NH98] D. Nessett and P. Humenn. The Multilayer Firewall. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, pages 13–27, March 1998.

[NIS94] Digital Signature Standard, May 1994.

[NS78] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–998, December 1978.

[Pos81] Jon Postel. Internet Protocol. Internet RFC 791, 1981.

[Pre96] V. Prevelakis. *A Model for the Organisation and Dynamic Reconfiguration of Information Networks*. PhD thesis, University of Geneva, Switzerland, 1996.

[PS99] J.S. Park and R. Sandhu. RBAC on the Web by smart certificates. In *Proceedings of the 4th ACM Workshop on Role-Based Acess Control (RBAC)*, pages 1–9, October 1999.

[RRSW97] C. Rigney, A. Rubens, W. Simpson, and S. Willens. Remote Authentication Dial In User Service (RADIUS). Request for Comments (Proposed Standard) 2138, Internet Engineering Task Force, April 1997.

[SC98] L.A. Sanchez and M.N. Condell. Security Policy System. Internet draft, work in progress, Internet Engineering Task Force, November 1998.

[Sch96] B. Schneier. *Applied Cryptography*. John Wiley, 1996.

[Sch00] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2000.

[Sim95] William Simpson. IP in IP Tunneling. Internet RFC 1853, October 1995.

[Skl93] K. Sklower. A tree-based routing table for Berkeley Unix. Technical report, University of California, Berkeley, 1993.

[Sol] Solsoft web site. http://www.solsoft.com/.

[SP98] Ravi S. Sandhu and Joon S. Park. Decentralized user-role assignment for web-based intranets. In *ACM Workshop on Role-Based Access Control*, pages 1–12, 1998.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[SSB$^+$95] D.L. Sherman, D.F. Sterne, L. Badger, S.L. Murphy, K.M. Walker, and S.A. Haghighat. Controlling network communication with domain and type enforcement. In *Proceedings of the 18th National Information Systems Security Conference*, pages 211–220, October 1995.

[SWY01] K. E. Seamons, M. Winslett, and T. Yu. Limiting the Disclosure of Access Control Policies During Automated Trust Negotiation. In *Proceedings of the 2001 Network and Distributed System Security Symposium (SNDSS)*, pages 109–124, February 2001.

[Tec] Telcordia Technologies. Evaluating the size of the Internet. http://www.netsizer.com/.

[TKS01] J. Trostle, I. Kosinovsky, and M. M. Swift. Implementation of Crossrealm Referral Handling in the MIT Kerberos Client. In *Proceedings of the 2001 Network and Distributed System Security Symposium (SNDSS)*, pages 109–124, February 2001.

[TOB98] D. Thomsen, D. O'Brien, and J. Bogle. Role Based Access Control Framework for Network Enterprises. In *Proceedings of the 14th Annual Computer Security Applications Conference*, December 1998.

[TOP99] D. Thomsen, R. O'Brien, and C. Payne. Napoleon Network Application Policy Environment. In *Proceedings of the 4th ACM Workshop on Role-Based Acess Control (RBAC)*, pages 145–152, October 1999.

[Tsu92] G. Tsudik. Policy Enforcement in Stub Autonomous Domains. In *Lecture Note in Computer Science 648, ESORICS '92*, pages 229–257. Springer-Verlag, 1992.

[Ven92] W. Venema. TCP WRAPPER: Network monitoring, access control and booby traps. In *Proceedings of the Third USENIX UNIX Security Symposium*, pages 85–92, 1992.

[VJ99] Jan Vitek and Christian Jensen. *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1999.

[WABL94] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos Operating System. *TOCS*, 12(1):3–32, February 1994.

[WD01] A. Westerlund and J. Danielsson. Heimdal and Windows 2000 Kerberos — how to get them to play together. In *Proceedings of the 2001 USENIX Annual Technical Conference, Freenix Track*, pages 267–272, June 2001.

[Wiz] Network Wizards. Internet Domain Survey. http://www.isc.org/ds.

[Woo01] A. Wool. Architecting the Lumeta Firewall Analyzer. In *Proceedings of the 10th USENIX Security Symposium*, pages 85–97, August 2001.

[WVTP97] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high speed IP routing lookups. In *Proceedings of SIGCOMM '97*, pages 25–36, September 1997.