Trust Management for IPsec

Matt Blaze
AT&T Labs - Research, mab@research.att.com

John Ioannidis
AT&T Labs - Research, ji@research.att.com

Angelos D. Keromytis
Columbia University, angelos@cs.columbia.edu

IPsec is the standard suite of protocols for network-layer confidentiality and authentication of Internet traffic. The IPsec protocols, however, do not address the *policies* for how protected traffic should be handled at security endpoints. This paper introduces an efficient policy management scheme for IPsec, based on the principles of trust management. A *compliance check* is added to the IPsec architecture that tests packet filters proposed when new security associations are created for conformance with the local security policy, based on credentials presented by the peer host. Security policies and credentials can be quite sophisticated (and specified in the trust-management language), while still allowing very efficient packet-filtering for the actual IPsec traffic. We present a practical, portable implementation of this design, based on the KeyNote trust-management language, that works with a variety of Unix-based IPsec implementations. Finally, we discuss some applications of the enhanced IPsec architecture.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms: Security,Languages

Additional Key Words and Phrases: Credentials, IPsec, KeyNote, network security, policy, trust management.

1. INTRODUCTION

The IPsec protocol suite, which provides network-layer security for the Internet, has recently been standardized in the IETF and is beginning to make its way into commercial implementations of desktop, server, and router operating systems. For many applications, security at the network layer has a number of advantages over security provided elsewhere in the protocol stack. The details of network semantics are usually hidden from applications, which therefore automatically and transparently take advantage of whatever network-layer security services their environment provides. More importantly, IPsec offers a remarkable flexibility not possible at higher- or lower- layer abstractions: security can be configured end-to-end (protecting traffic between two hosts), route-to-route (protecting traffic passing

This work was supported by DARPA under Contract F39502-99-1-0512-MOD P0001. A previous version of this paper appeared as [Blaze et al. 2001].

over a particular set of links), edge-to-edge (protecting traffic as it passes between "trusted" networks via an "untrusted" one), or in any other configuration in which network nodes can be identified as appropriate security endpoints.

Despite this flexibility, IPsec does not itself address the problem of managing the *policies* governing the handling of traffic entering or leaving a host running the protocol. By itself, the IPsec protocol can protect packets from external tampering and eavesdropping, but does nothing to control which hosts are authorized for particular kinds of sessions or to exchange particular kinds of traffic. In many configurations, especially when network-layer security is used to build firewalls and virtual private networks, such policies may necessarily be quite complex. There is no standard interface or protocol for controlling IPsec tunnel creation, and most IPsec implementations provide only rudimentary, packet-filter-based and ACL-based policy mechanisms.

The crudeness of IPsec policy control, in turn, means that in spite of the availability of network-layer security, many applications are forced to duplicate at the application or transport layer cryptographic functions already provided at the network layer.

There are three main contributions in this paper: we introduce a new policy management architecture for IPsec, based on the principles of trust management; we present a design that integrates this architecture with the KeyNote Trust Management system; finally, we present a practical, portable implementation of this design, currently distributed in open-source form in OpenBSD.

1.1 IPsec Packet Filters and Security Associations

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, as introduced in [Ioannidis and Blaze 1993] and standardized in [Kent and Atkinson 1998b]. 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.

Let us examine the security policy decisions an IPsec processor must make. When we discuss "policy" in this paper, we refer specifically to the network-layer security policies that govern the flow of traffic among networks, hosts, and applications. Observe that policy must be enforced whenever packets arrive at or are about to leave a network security endpoint (which could be an end host, a gateway, a router, or a firewall).

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).

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?
  —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[Harkins and Carrel 1998].

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.*[McCanne and Jacobson 1993].

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.

A further issue with IPsec policy control is the need for two hosts to discover and negotiate the kind of traffic they are willing to exchange. When two hosts governed by their own policies want to communicate, they need some mechanism for determining what, if any, kinds of traffic the combined effects of one another's policies are permitted. Again, IPsec itself does not provide such a mechanism; when a host attempts to create an SA, it must know in advance that the policy on the remote host will accept it. The operation then either succeeds or fails. While this may be sufficient for small VPNs and other applications where both peers are under the same administrative control, it does not scale to larger-scale applications such as public servers.

1.2 Related Work

The IKE specification [Harkins and Carrel 1998] makes use of the Subject Alternate Name field in X.509 [CCITT 1989; Housley et al. 1999] certificates to encode the packet selector the certificate holder may use during IKE Quick Mode. Beyond this, no standard way has yet been defined for negotiating, exchanging, and otherwise handling IPsec security policy.

[Sanchez and Condell 1998] 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.

[Condell et al. 1999] describes a language for specifying communication security policies, heavily oriented toward IPsec and IKE. SPSL is based on the Routing Policy Specification Language (RPSL) [Alaettinoglu et al. 1998]. 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. While it is possible to translate from SPSL policy statements directly to KeyNote policies (and vice versa, for local host policies), it is not possible to express trust relationships and refine the authority conferred to a principal (another administrator or an end-user) in SPSL alone.

A number of other Internet Drafts have been published defining various directory schemata for IPsec policy. Similar directory-based work has also started in the context of the IETF Policy Framework Working Group. It is still too early to determine what the results of that effort will be.

COPS [Boyle et al. 2000] defines a simple client/server protocol wherein a Policy Enforcement Point (PEP) communicates with a Policy Decision Point (PDP) in order to determine whether a requested action is permissible. COPS is mostly oriented toward admission control for RSVP [Braden et al. 1997] or similar protocols. It is not clear what its applicability to IPsec security policy would be.

RADIUS [Rigney et al. 1997] and its proposed successor, DIAMETER [Calhoun et al. 1999], 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.

We first proposed the notion of using a trust management system for network-layer security policy control in [Blaze et al. 1999].

2. TRUST MANAGEMENT FOR IPSEC

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 could be based on a public-key or Needham-Schroeder [Needham and Schroeder 1978] 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. Recall that policy control is a central motivation 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.

—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.

The *trust-management* approach [Blaze et al. 1996] for checking compliance with security policy provides exactly the interface and abstractions required.

2.1 The KeyNote Trust Management System

Because we make extensive use of the concepts of trust management, and especially the KeyNote language, we provide a brief review of those concepts here.

The notion of *trust management* was introduced in [Blaze et al. 1996]. A trust-management system provides a standard interface that applications can use to test whether potentially dangerous actions comply with local security policies.

More formally, trust-management systems are characterized by:

—A method 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 for determining how an action requested by principals should be handled, given a policy and a set of credentials.

KeyNote is a simple and flexible trust-management system designed to work well for a variety of applications. In applications using KeyNote, policies and credentials are written in the same language. The basic unit of KeyNote programming is the *assertion*. Assertions contain programmable predicates that operate on the requested attribute set and limit the actions that principals are allowed to perform. KeyNote assertions are small, highly-structured programs. Authority can be delegated to others; a digitally signed assertion can be sent over an untrusted network and serve the same role as traditional certificates. Unlike traditional policy systems, policy in KeyNote is expressed as a combination of *unsigned* and *signed* policy assertions (signed assertions are also called credentials). There is a wide spectrum of possible combinations; on the one extreme, all system policy is expressed in terms of local (unsigned) assertions. On the other extreme, all policy is expressed as signed assertions, with only one rule (the root of the policy) being an unsigned assertion that delegates to one or more trusted entities. The integrity of each signed assertion is guaranteed by its signature; therefore, there is no need for these to be stored within the security perimeter of the system.

KeyNote allows the creation of arbitrarily sophisticated security policies, in which entities (which can be identified by cryptographic public keys) can be granted limited authorization to perform specific kinds of trusted actions.

When a "dangerous" action is requested of a KeyNote-based application, the application submits a description of the action along with a copy of its local security policy to the KeyNote interpreter. Applications describe actions to KeyNote with a set of attribute/value pairs (called an *action attribute set* in KeyNote terminology) that describe the context and consequences of security-critical operations. KeyNote then "approves" or "rejects" the action according to the rules given in the application's local policy.

KeyNote assertions are written in ASCII and contain a collection of structured fields that describe which principal is being authorized (the *Licensee*), who is doing the authorizing (the *Authorizer*) and a predicate that tests the action attributes (the *Conditions*). For example:

```
Authorizer:  "POLICY"
Licensees:  "Boris Yeltsin"
Conditions:
 EmailAddress == "yeltsin@kremvax.ru"
```

means that the "POLICY" principal authorizes the "Boris Yeltsin" principal to do any action in which the attribute called "EmailAddress" is equal to the string "yeltsin@kremvax.ru". An action is authorized if assertions that approve the action can link the "POLICY" principal with the principal that authorized the action. Principals can be public keys, which provides a natural way to use KeyNote to control operations over untrustworthy networks such as the Internet.

A complete description of the KeyNote language can be found in [Blaze et al. 1999].

2.2 KeyNote Control for IPsec

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.

—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 architecture divides 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.

2.3 Policy Discovery

While the IPsec compliance-checking model described above can be used by itself to provide security policy support for IPsec, there are two additional issues that need to be addressed if such an architecture is to be deployed and used.

Recall that in a trust management system such as KeyNote, the rules governing any given request might be contained not only in a local policy file but might also be contained in signed credentials, which, because they are signed, could be stored anywhere. If a credential containing a relevant clause is not available to the machine doing the compliance checking, a "legal" action might not be approved.

It is therefore important to discover and acquire all the credentials required to approve an action. Although users or hosts may be expected to manage locally policies and credentials that directly refer to them, they may not know of intermediate credentials (*e.g.,* those issued by administrative entities) that may be required by the hosts with which they want to communicate. Consider the case of a large organization, with two levels of administration; local policy on the firewalls trusts only the "corporate security" key. Users obtain their credentials from their local administrators, who authorize them to connect to specific firewalls. Thus, one or more intermediate credentials delegating authority from corporate security to the various administrators is also needed if a user is to be successfully authorized. Naturally, in more complex network configurations (such as extranets) multiple levels of administration may be present. Some method for determining what credentials are relevant and how to acquire them is needed. The most straightforward solution is top-to-bottom distribution of credentials: that is, each administrator passes to the next level (be it end-users or another administration level) the necessary credentials it received from the higher level, along with the newly-minted credentials. While this approach is simple and does not require any additional provisioning, it does not work well if the various credentials in a "bundle" expire at different times: the end-user still needs some way of acquiring a fresh version of expired credentials.

One way of rectifying this is to have the host that intends to initiate an IKE exchange use a simple protocol, which we call Policy Query Protocol (PQP), to acquire or update credentials relevant to a specific intended IKE exchange. The initiator presents a public key to the responder and asks for any credentials where the key appears in the Licensees field. By starting from the initiator's own key (or from some key that delegates to the initiator), it is possible to acquire all credentials that the responder has knowledge of that may be of use to the initiator. The responder can also provide pointers to other servers where the initiator may find relevant credentials; in fact, the responder can just provide a pointer to some other server that holds credentials for an administrative domain.

Since the credentials themselves are signed, there is no need to provide additional security guarantees in the protocol itself. However, any local policies that the responder discloses would have to be signed prior to being sent to the initiator; the fact that a KeyNote policy "becomes" a credential simply by virtue of being signed is very useful here. Also, the PQP server can have its own policy concerning which hosts are allowed to query for credentials.

As described, the PQP protocol does not use encryption; thus, an eavesdropper could collect potentially sensitive information encoded in the credentials (*e.g.,* topology and services ran behind a firewall, privileges of a user, *etc.*). Moreover, an active attacker could determine all the privileges of a key simply by running the PQP protocol against a firewall or credential repository. To avoid this, the credentials sent to the client can be encrypted under the first public key provided in a PQP session; thus, only the key holder can see these credentials. This approach works well with credential servers, where the credentials can be pre-encrypted and simply served to clients. For firewalls, or where the pre-computation or storage requirements cannot be met, PQP can be ran over an IPsec-protected session; the firewall or credential server is configured to accept IPsec SAs that can only transfer traffic from the client to the server's PQP port; the PQP daemon can then extract the public key used for the client authentication in IKE, and compare it to the initial key used in the PQP session.

The second problem is determining our own capabilities based on the credentials we hold. This is in some sense complementary to compliance checking; by analyzing our credentials in the context of our peer's policy, it is possible to determine what types of actions are accepted by that peer. That is, we can discover what kinds of IPsec SA proposals are accepted by a remote IKE daemon. This can assist in avoiding unnecessary IKE exchanges (if it is known in advance that no SAs acceptable by both parties can be agreed upon), or narrow down the set of proposals we send to our peer. Note that if a host reveals all the relevant credentials and policies using the Policy Query Protocol, another host can determine in advance and off-line exactly what proposals that host will accept.

When the firewall policies are deemed too sensitive for disclosure, the responder can either insist on running PQP over IPsec (allowing for release of such policies only to trusted clients), or can simply not reveal any sensitive information. In the latter case, the capability determination process will compute capabilities based on partial information; at worse, this can cause the initiation of IKE exchanges that will ultimately fail (which, in any case, is the current state of affairs). Two factors work in our favor here: first, the "last" credential (that issued by the last administrator in the hierarchy to an end-user) typically contains enough information to determine many of the acceptable parameters for an exchange (for example, see Figure 8); second, since these credentials are targeted to specific users, revealing these does not have a significant impact in revealing the overall system or network policy. Lastly, there has been some recent work in the area of regulated policy release[Bonatti and Samarati 2000] which may be relevant here, although we have not yet investigated its applicability.

Credential composition is a fairly straightforward, if potentially expensive, operation: we start by constructing a graph from the peer's policy to our key. We then reduce each clause in the Conditions field of each credential to its Disjunctive Normal Form (DNF). This can be an expensive operation: in the worst case scenario, when a Conditions clause is in Conjunctive Normal Form (CNF) (as it is in Figure 8), the algorithmic cost of this conversion can be quadratic in the number of terms in the clause. While this would be prohibitive for large policies, it is envisioned that most of the credentials would be fairly situation-specific, and thus small in size and number of terms. Furthermore, it is possible to create the credentials such that they are in DNF form to begin with, especially when some higher-level tool, such as a GUI or a script, is used to generate the credentials.

To determine the authorization in a chain of two credentials, we need to compute the intersection of their authorizations. This is a linear-cost operation over the number of terms in the DNF expressions of the two credentials. For larger chains (or, indeed, arbitrary graphs of credentials), we can apply the same algorithm recursively. At the end of this operation, we have a list of acceptable proposals, which the IKE daemon can then use to construct valid SA proposals for the remote host.

Note that this operation is typically done by the initiator, and thus has no significant performance impact on the responder, which may be a busy security gateway.

3. IMPLEMENTATION

To demonstrate our policy management scheme, we implemented the architecture described in the previous section within the OpenBSD IPsec stack [Keromytis et al. 1997; Hallqvist and Keromytis 2000]. OpenBSD's IKE implementation (called `isakmpd`) supports both passphrase and X.509 certificate authentication. We modified `isakmpd` to use KeyNote instead of the configuration-file based mechanism that was used to validate new Security Associations. A host's local policy is given in a text file (`/etc/isakmpd.policy`) that contains KeyNote policy assertions.

3.1 The OpenBSD IPsec Architecture

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

Outgoing packets are processed in the `ip_output()` routine. The Security Policy Database (SPD)[1] 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[2], 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 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.

3.2 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 in-

[1]The SPD is part of all IPsec implementations[Kent and Atkinson 1998b], 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).

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

formation 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, Perfect Forward Secrecy (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 the appendix. For the full list, consult the OpenBSD manual pages.

3.3 Policies for Passphrase Authentication

If passphrases are used as the IKE authentication method, KeyNote policy control may be used to directly authorize the holders of the passphrases. Passphrases are encoded as KeyNote principals by taking the ASCII string corresponding to the passphrase prefixed with the string "passphrase:" Thus, the following policy would allow anyone knowing the passphrase "foobar" to establish an SA with the ESP [Kent and Atkinson 1998a] protocol.

```
Authorizer:  "POLICY"
Licensees:  "passphrase:foobar"
Conditions:
  app_domain == "IPsec Policy"
  && esp_present == "yes" ;
```

```
Authorizer: "POLICY"
Licensees: "DN:/CN=Certification Authority Foo/Email=ca@foo.com"
Conditions:  ...
```

Fig. 1. Sample credential with X.509 DN as Licensee

Using the `passphrase:` tag requires policies to be kept private. To avoid this, a hashed version of the passphrase may be used instead (*e.g.,* using the `passphrase-sha1-hex:` prefix). In the previous example, this would be:

```
passphrase-sha1-hex:8843d7f92416211de9ebb963ff4ce28125932878
```

Since there are no restrictions on the length or format of the passphrases used, salting (as is common in password schemes, to protect against dictionary attacks) is not strictly necessary. However, salting can be used to increase the cost of dictionary attacks against passphrases encoded in this manner.

3.4 Policies for X.509-based Authentication

More interesting is the interaction between KeyNote policy and X.509 public-key certificates for authentication. Most IKE implementations (including ours) allow the use of X.509 certificates for authentication. Furthermore, there exist a number of commercial tools that let administrators manage large collections of users using X.509. Allowing for interoperability with these implementations is a good test of our architecture and can make transition to a KeyNote-based infrastructure considerably smoother.

Implementing this interoperability is straightforward: KeyNote policies may be used to delegate directly to X.509 certificates. The principals specified may be the certificates themselves (in pseudo-MIME format, using the `x509-base64:` prefix), the subject public key, or the Subject Canonical Name. An example is given in Figure 1.

For each X.509 certificate received and verified as part of an IKE exchange, an *ad hoc* KeyNote credential is generated. This credential maps the Issuer/Subject keys of the X.509 certificate (from the respective fields) to Authorizer/Licensees keys in KeyNote. Thus, as chains of X.509 certificates are formed during regular operation, corresponding chains of KeyNote credentials are formed. This allows policies to delegate to a CA, and have the same restrictions apply to all users certified by that CA. Specific users may be granted more privileges by direct authorization in the host's policy.

3.5 Policies for KeyNote Credentials

KeyNote credentials may be passed directly during the IKE exchange, in the same manner as X.509 certificates[3]. This method offers the most flexibility in policy specification, as it allows principals to further delegate authority to others through arbitrarily complex graphs of authorization. Any signed KeyNote credentials received during the IKE exchange are passed to the KeyNote interpreter directly as part of the query.

[3]More specifically, they are passed via the IKE CERT(IFICATE) payload.

KeyNote credentials are especially useful in the remote administration case, where the policies of many IPsec endpoints are controlled by a central administrator. Here, the policy of each host would delegate all authority to the public key of the central administrator. The administrator would then issue credentials that contain the details of the policy under which they were issued. These credentials are presented as part of each IKE exchange by any host requesting access. This eliminates the need to update large numbers of machines as the details of organizational policies change. Adding a new host is accomplished by having the administrator issue a new credential for that host; that host may then use the newly-issued credential to communicate with any other host that obeys the above policy. No policy changes are necessary to these hosts. Revoking access to a host is implemented through short-lived credentials. New credentials are made available periodically through a WWW or FTP server; clients can download them from there, without any security implications (since the credentials are signed, their integrity is guaranteed). If credential confidentiality is an issue, these credentials could be encrypted with the public key of the user before they are made available.

Regardless of the authentication method in use, `isakmpd` calls KeyNote to determine whether each proposed SA should be established. After taking into consideration policies, credentials, and the attributes pertinent to the SA, KeyNote returns a positive or negative answer. In the former case, the protocol exchange is allowed to proceed as usual. In the latter, an informational message is sent to the remote IKE daemon and the exchange is dropped. Note that, if an administrator were to manually establish SPD rules (by directly manipulating the SPD), KeyNote and the SPD might disagree; in that case, no SA would ever be established and no packets would be sent out for that communication flow (since the SPD would require an SA).

The basic data flows for KeyNote-controlled IPsec input and output processing are given in Figures 2 and 3, respectively.

Input processing begins with a packet arriving at a network interface (#1 in Figure 2). The Security Policy Database is consulted (#2) and one of three actions is followed. If the packet is an IPsec packet, it is sent (#3a) to the IPsec processing code, which will consult the SA Database (#11) to process the packet; the decapsulated packet is then fed back to the IP input queue (#12). If the SPD says that the packet should just be accepted, it is sent (#3b) to the corresponding higher-layer protocol, or forwarded, as appropriate. If the SPD says that the packet should be dropped, no further processing is done. Otherwise (#3c), the Security Association setup process is triggered. The SA Database is consulted (#4); if an SA is found there, the packet is dropped because it should have already been sent as an IPsec packet (and it was not, or path #3a would have been followed). Next, the Policies and Credentials database is consulted (#5); this is done by calling the KeyNote interpreter, supplying it the relevant details of the packet (addresses, protocol, ports, *etc.*). The KeyNote interpreter, in turn, consults *its* database of policies and credentials, and determines whether the packet should be just accepted, dropped, or needs IPsec protection. If the latter is the case, the IKE daemon is triggered (#6). It establishes SAs with its peer (#7), during which process it will also need to consult the policy and credentials database (#8), and may also update it with additional credentials acquired during the IKE exchange. The SA and

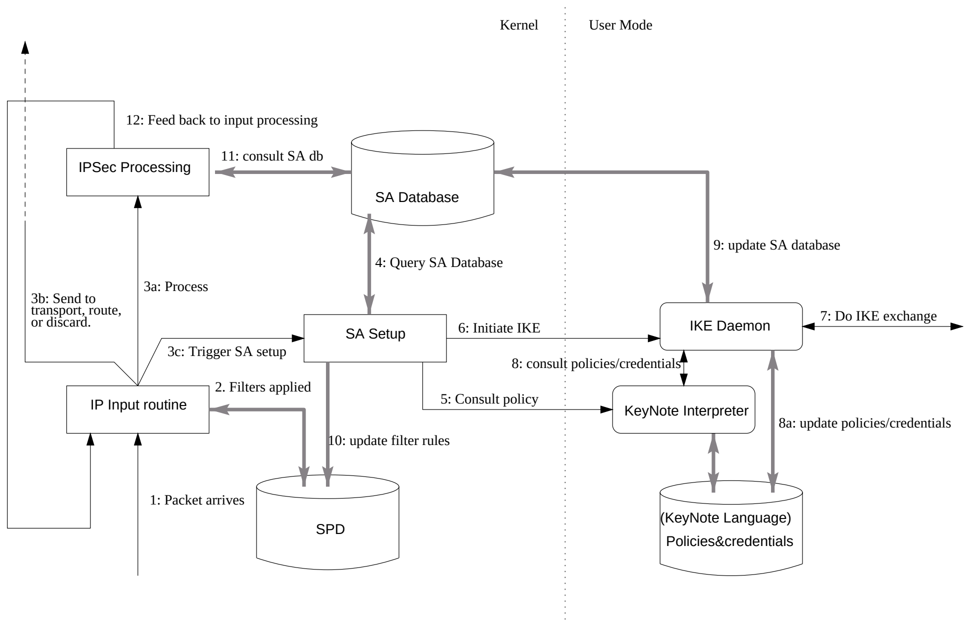

Fig. 2. KeyNote-Controlled IPsec Input Processing

SPD Databases are then updated (#9, #10) as necessary based on the information negotiated by IKE. The unprotected packet that triggered the SA establishment is dropped.

Output processing starts when a packet arrives (#1 in Figure 3) at the IP output code from either a higher-level protocol or from the forwarding code. The Security Policy Database is consulted (#2) to determine whether the packet should be protected with IPsec or not; if no protection is needed, the packet is simply sent out (#3a). Otherwise, it is sent to the IPsec processing code (#3b). A lookup (#4) in the SA database determines whether an SA for this packet already exists; if so, the appropriate transforms are applied and the resulting packet is output (#5a). If an SA did not exist, the SA setup process is invoked (#5b). The system policy (as contained in the SPD) is consulted (#6), and if policy relevant to this packet is found, the IKE exchange is triggered (#7), otherwise the packet is simply dropped. During the IKE exchange (#8), the local policy and credentials are consulted (#9), and any credentials fetched from the peer during the exchanged are subsequently stored (#10) in the local database. If the IKE exchange results in SAs being created, these are stored back in the SA database (#11). Finally, the SPD is updated (#12) if necessary, and subsequent packets can be processed (the original unprotected packet is dropped).

It should be obvious from the above that, in our architecture, the SPD has become a policy cache; the "real" policy is expressed in terms of KeyNote assertions and credentials. There are two ways of populating the cache. The first, described above, is to populate it on-demand. If a filter rule does not exist in the SPD, KeyNote is invoked to determine what should be done with the packet; based on the response

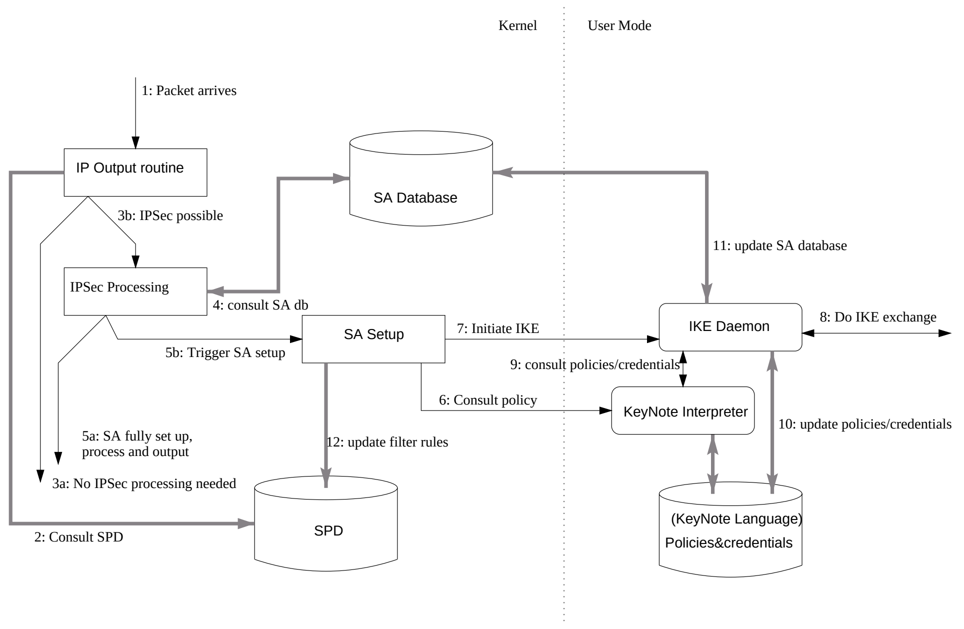

Fig. 3. KeyNote-Controlled IPsec Output Processing

from KeyNote, a rule is installed in the SPD that makes further KeyNote queries unnecessary. The second approach is to analyze all policies at startup time and populate the SPD accordingly. This avoids the cost of a cross-domain call (from the kernel to a user-space policy daemon) per cache miss, but requires re-initialization of the SPD every time the policy changes.

3.6 Policy Updates

Changing policy in the simple case is straightforward: the new policies are placed in `isakmpd.policy`. When existing IPsec SAs expire and are subsequently re-negotiated, or when new IPsec SAs are established, the new policy will automatically be taken into consideration. If we want new policy to be applied to existing IPsec SAs, we can simply examine the existing SAs in the context of the new policy, pretending we are now establishing them. If the updated policy permits the old SAs, no further action is required; otherwise, they are deleted.

3.7 Performance

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 Pentium II at 450Mhz. Because evaluating the base KeyNote policies themselves does not require the verification of digital signatures, the KeyNote compliance check is generally very fast: with a small number of policy assertions, initialization and verification overhead is approximately 130 microseconds. This number increases linearly with the size and the number of policy assertions that are actually evaluated, each such assertion adding approximately 20 microseconds. The generation of the shadow delegation tree is also very low cost. When using KeyNote credentials for both authentication and policy specification, the cost of public-key signature verification is incurred. This cost is identical to that of the standard X.509 case (and indeed to that of any other public-key authentication mechanism). Signatures in KeyNote credentials are verified as needed and only the first time they are used — the verification result is cached and reused. Credential expiration is handled by the general KeyNote processing, as part of the Conditions field; thus policies and credentials that have expired do not contribute in authorizing an SA and no special handling is needed. In all cases, the cost of KeyNote policy processing is several orders of magnitude lower than the cost of performing the public-key operations that it is controlling.

KeyNote policy control contributed only a negligible increase in the code size of the OpenBSD IPsec implementation. To add KeyNote support to *isakmpd* we added about 1000 lines of "glue" code to `isakmpd`. Almost all of this code is related to data structure management and 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.

4. OTHER APPLICATIONS OF TRUST MANAGEMENT IN IPSEC

In this section we illustrate some uses of the enhanced IPsec architecture we presented.

4.1 The Distributed Firewall

In [Ioannidis et al. 2000], the work presented in this paper implements a "distributed firewall". Distributed firewalls overcome several significant shortcomings of traditional firewalls, which depend on artificial constraints in network topology to enforce centrally-specified security policies:

—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 been and likely will continue to outpace Moore's Law [Dahlin 1995].

—The assumption, inherent in traditional firewall deployment, that all insiders are equally trusted 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, telecommuting, *etc.*). Consequently, the traditional notion of a security perimeter can no longer hold unmodified; for example, it is desirable that telecommuters' systems comply with the corporate security policy.

—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.

—End-to-end encryption can also be a threat to firewalls [Bellovin 1999], 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 to the users on behalf of 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.

Despite these shortcomings, firewalls offer the advantage of enforcing centrally-defined policy. Thus, the distributed firewall work attempts to address all these problems while allowing centralized policy specification. The distributed firewall uses IPsec as a means for distributing policies, in the form of KeyNote credentials, for various network applications; these policies are enforced by individual end-hosts and servers, thus overcoming most of the problems outlined above. Enforcement is done in a TCP-connection granularity, although more recently this has been extended down to individual packets without unduly affecting performance.

The prototype system was implemented using OpenBSD, and is comprised of three parts: a set of kernel extensions, which implement the enforcement mechanism, 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. The prototype implementation is approximately 1150 lines of *C* code, equally split among the three components.

Figure 4 shows a graphical representation of the system, with all its components. For more details, see [Ioannidis et al. 2000].

4.2 STRONGMAN

The STRONGMAN architecture [Keromytis 2001] ties together multiple security policy mechanisms within a single system image. The architecture supports many application-specific policy languages, and automatically distributes and uniformly enforces the single security policy across all enforcement points. Furthermore, STRONGMAN allows enforcement points to be chosen appropriately to meet both security and performance requirements.

Other concerns STRONGMAN addresses are:

—Policy updates must be as cheap as possible, since these are common and often-used operations in any system (adding/giving privileges to a user, removing/revoking privileges from a user).

—Security policies for a particular application should be specified in an application-specific language, and a single specification should be able to control the behavior of any needed security mechanism.

—Finally, administrators should be able to independently specify policies over their own domain: this should be true whether the administrator manages particular applications within a security domain, or manages a subdomain of a larger administrative domain.

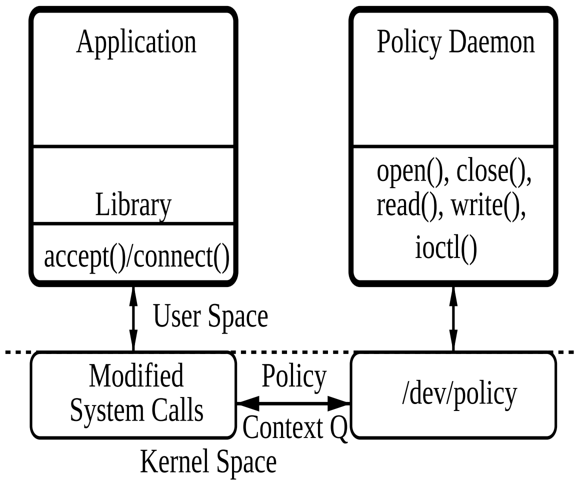

Fig. 4. The Figure shows a graphical representation of the system, with all its components. The core of the enforcement mechanism lives in kernel space and is comprised of the two modified system calls that interest us, `connect(2)` and `accept(2)`. The policy specification and processing unit lives 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*.

The architecture can be described at the highest level by Figure 5.

At the lowest level, the system supports "lazy instantiation" of policy on the enforcement points in order to minimize the resources consumed by policy storage. In other words, an enforcement point should only learn those parts of its policy that it actually has to enforce as a result of user service access patterns. A further benefit of this approach is that policy may be treated as "soft state," and thus be discarded by the enforcement point when resources are running low and recovered when space permits or after a crash.

STRONGMAN shifts as much of the operational burden as possible to the end users' systems because enforcement points are generally overloaded with processing requests and mediating access. As an example, in the context of "lazy policy instantiation" described above, the users' systems can be made responsible for acquiring the policies that apply to the users and for providing these to the enforcement points.

There is a distinction between high and low level policy in STRONGMAN; in particular, there may be multiple high-level policy specification mechanisms (different languages, GUIs, *etc.*), all translating to the same lower-level policy expression language. A powerful, flexible, and extensible low-level mechanism that is used as a common "policy interoperability layer" allows us to use the same policy model across different applications, without mandating the use of any particular policy front-end. 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*. In our prototype system, KeyNote serves the role of

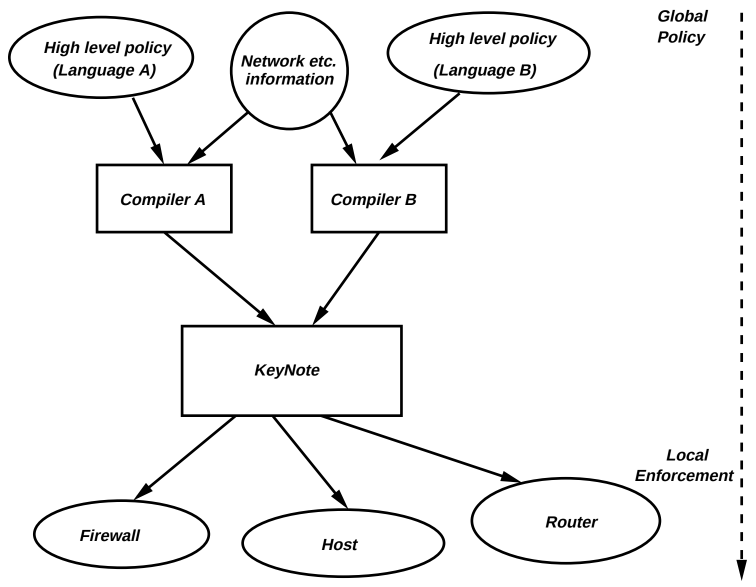

Fig. 5. KeyNote used as a policy interoperability layer. Policy composition in STRONGMAN does not depend on using the same compiler to process all the high-level policies.

the policy interoperability layer.

Finally, using KeyNote as the low-level policy system allows for decentralized and hierarchical management and supports privilege delegation to other users. Note that delegation allows any user to be treated as an "administrator" of her delegates; conversely, administrators in such a system can simply be viewed as users with very broad privileges. This permits both decentralized management (different administrators/users are made responsible for delegating and potentially refining different sets of privileges), and collaborative networking (by treating the remote administrator as a local user with specific privileges she can then delegate to her users). Limited privileges can be conferred to administrators of other domains, who can then delegate these to their users appropriately; this allows for Intranet-style collaborations.

5. CONCLUSIONS, FUTURE WORK, AVAILABILITY

We have demonstrated a practical and useful approach to managing trust in network-layer security. One of the most valuable features of trust management for IPsec SA policy management is its handling of policy delegation, which essentially unifies remote administration with credential distribution.

Perhaps the most important contribution of this work is our use of a two level policy specification hierarchy to control IPsec traffic. At the packet level, we use a specialized, very efficient, but less expressive filtering language that provides the basic control of traffic through the host. The installation of these packet filters, in turn, is controlled by a more expressive, general purpose, but less efficient trust-management language. Our performance measurements provide encouraging evidence that this approach is quite viable, providing a very high degree of control over traffic without the performance impact normally associated with highly expressive, general purpose mechanisms. It is possible that this approach has merit in applications beyond controlling network-layer security.

Because the KeyNote language on which this work is based is application-independent, our scheme can be used as the basis for a more comprehensive policy management architecture that ties together different aspects of network security with policies for IPsec and packet filtering. For example, a general network security policy might specify the acceptable mechanisms for remote access to a private corporate network over the Internet; such a policy may, for example, allow the use of clear-text passwords only if traffic is protected with IPSEC or some transport-layer security protocol (*e.g.,* SSH [Ylonen et al. 1999]). Multi-application policies would, of course, require embedding policy controls into either an intermediate security enforcement node (such as a firewall) or into the end applications and hosts [Ioannidis et al. 2000]. This approach is the subject of ongoing research.

The KeyNote trust-management system is available in an open source toolkit; see the KeyNote web page at

`http://www.crypto.com/trustmgt/`

for details. The KeyNote IPsec trust-management architecture is distributed with OpenBSD 2.6 (and later), which is available from

`http://www.openbsd.org/`

Because the policy management functionality is implemented entirely in the user-level `isakmpd`, the system is readily portable to other IPsec platforms (especially those based on BSD implementations).

ACKNOWLEDGMENTS

The distributed firewall is work done by Sotiris Ioannidis, Angelos Keromytis, Jothathan Smith (University of Pennsylvania), and Steve Bellovin (AT&T Labs – Research). The STRONGMAN architecture is the subject of Angelos Keromytis' PhD thesis, and Sotiris Ioannidis, Michael Greenwald, and Jonathan Smith (University of Pennsylvania) have all collaborated and contributed to that work.

REFERENCES

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

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

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

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

Blaze, M., Ioannidis, J., and Keromytis, A. 1999. Trust Management and Network Layer Security Protocols. In *Proceedings of the 1999 Cambridge Security Protocols International Workshop* (1999).

BLAZE, M., IOANNIDIS, J., AND KEROMYTIS, A. 2001. Trust Managent for IPsec. In *Proc. of Network and Distributed System Security Symposium (NDSS)* (February 2001), pp. 139–151.

BONATTI, P. AND SAMARATI, P. 2000. Regulating Service Access and Information Release on the Web. In *Proc. of the Seventh ACM Conference on Computer and Communications Security* (Athens, Greece, 2000).

BOYLE, J., COHEN, R., DURHAM, D., HERZOG, S., RAJAN, R., AND SASTRY, A. 2000. The COPS (Common Open Policy Service) Protocol. Request for comments (proposed standard) (January), Internet Engineering Task Force.

BRADEN, R., ZHANG, L., BERSON, S., HERZOG, S., AND JAMIN, S. 1997. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. Internet RFC 2208.

CALHOUN, P., RUBENS, A., AKHTAR, H., AND GUTTMAN, E. 1999. DIAMETER Base Protocol. Internet Draft (Dec.), Internet Engineering Task Force. Work in progress.

CCITT. 1989. *X.509: The Directory Authentication Framework*. Geneva: International Telecommunications Union.

CONDELL, M., LYNN, C., AND ZAO, J. 1999. Security Policy Specification Language. Internet draft (July), Internet Engineering Task Force.

DAHLIN, M. 1995. *Serverless Network File Systems*. Ph. D. thesis, University of California, Berkeley.

HALLQVIST, N. AND KEROMYTIS, A. D. 2000. Implementing Internet Key Exchange (IKE). In *Proceedings of the Annual USENIX Technical Conference, Freenix Track* (June 2000), pp. 201–214.

HARKINS, D. AND CARREL, D. 1998. The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409 (Nov.), Internet Engineering Task Force.

HOUSLEY, R., FORD, W., POLK, W., AND SOLO, D. 1999. Internet X.509 public key infrastructure certificate and CRL profile. Request for Comments 2459 (Jan.), Internet Engineering Task Force.

IOANNIDIS, J. AND BLAZE, M. 1993. The Architecture and Implementation of Network-Layer Security Under Unix. In *Fourth Usenix Security Symposium Proceedings* (October 1993). USENIX.

IOANNIDIS, S., KEROMYTIS, A., BELLOVIN, S., AND SMITH, J. 2000. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS) 2000* (November 2000), pp. 190–199.

KENT, S. AND ATKINSON, R. 1998a. IP Encapsulating Security Payload (ESP). Request for Comments (Proposed Standard) 2406 (Nov.), Internet Engineering Task Force.

KENT, S. AND ATKINSON, R. 1998b. Security Architecture for the Internet Protocol. Request for Comments (Proposed Standard) 2401 (Nov.), Internet Engineering Task Force.

KEROMYTIS, A. D. 2001. *STRONGMAN: A Scalable Solution to Trust Management in Networks*. Ph. D. thesis, University of Pennsylvania.

KEROMYTIS, A. D., IOANNIDIS, J., AND SMITH, J. M. 1997. Implementing IPsec. In *Proceedings of Global Internet (GlobeCom) '97* (November 1997), pp. 1948 – 1952.

MCCANNE, S. AND JACOBSON, V. 1993. A BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of USENIX Winter Technical Conference* (San Diego, California, Jan. 1993), pp. 259–269. Usenix.

NEEDHAM, R. AND SCHROEDER, M. 1978. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM 21*, 12 (December), 993–998.

RIGNEY, C., RUBENS, A., SIMPSON, W., AND WILLENS, S. 1997. Remote Authentication Dial In User Service (RADIUS). Request for Comments (Proposed Standard) 2138 (April), Internet Engineering Task Force.

SANCHEZ, L. AND CONDELL, M. 1998. Security Policy System. Internet draft, work in progress (November), Internet Engineering Task Force.

YLONEN, T., KIVINEN, T., SAARINEN, M., RINNE, T., AND LEHTINEN, S. 1999. SSH Protocol Architecture. Internet Draft (Feb.), Internet Engineering Task Force. Work in progress.

```
Authorizer: "POLICY"
Licensees: "passphrase:pedomellonaminno"
Conditions: app_domain == "IPsec policy"
  && doi == "ipsec" && pfs == "yes"
  && esp_present == "yes" && esp_enc_alg != "null"
  && remote_filter == "135.207.000.000-135.207.255.255"
  && local_filter == "198.001.004.0-198.001.004.255"
  && remote_ike_address == "198.001.004.001" ;
```

Fig. 6. Policy for Firewall of 135.207.0.0/16 Network.

Appendix 1: 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).

Appendix 2: Configuration Examples

Example 1: Setting up a VPN

In this example, two sites are connected over an encrypted tunnel. The authentication is done by a simple passphrase. The policy in Figure 6 is present at one of the firewalls. It specifies that packets between the 135.207.0.0/16 range of addresses and the 198.1.4.0/24 range of addresses have to be protected by ESP using encryption. The remote gateway, with which IKE will negotiate, is 198.1.4.1.

```
Authorizer: POLICY
Licensees: RAS_ADMIN_Key
Comment: delegate authority to a Remote Access administrator.
Local-Constants:
           RAS_ADMIN_Key_A = "rsa-base64:MDgCMQDMiEBn89VCSR3ajxr0bNRC\
                 Audlz5724fUaW0uyi4r1oSq8PaSC2v9QGS+phGEahJ8CAwEAAQ=="
Conditions: app_domain == "IPsec policy"
         && doi == "ipsec"
         && pfs == "yes"
         && ah_present == "no"
         && esp_present == "yes"
         && esp_enc_alg == "3des" && esp_auth_alg == "hmac-sha"
         && esp_encapsulation == "tunnel"
         && local_filter == "139.091.000.000-139.91.255.255"
         && remote_ike_address == remote_filter ;
```

Fig. 7. Mobile host local policy.

Example 2: Remote Access

Authority to allow remote access through the site firewall is controlled by several security officers, each one of whom is identified by a public key. A policy entry such as the one shown in Figure 7 exists for each individual security officer, and is stored in the `isakmpd` configuration file of the firewall. Note the last line in the Conditions field, which restricts remote users to negotiate only host-to-firewall SAs, without placing any restrictions to their actual address otherwise.

Each portable machine that is to be allowed in must hold a credential similar to that shown in Figure 8; the credential is signed by a security administrator. When weak encryption is used, the user can only read and send e-mail; when strong encryption is used, all kinds of traffic are allowed. During the IKE exchange, the user's `isakmpd` provides this credential to the firewall, which passes it on to KeyNote. The policy and the credential, taken together, express the overall access policy for the holder of key JIK. A similar policy (and a corresponding credential) is issued to the user (and firewall), to authorize the reverse direction (the firewall needs to prove to the user that it is authorized by the administrator to handle traffic to the 139.91.0.0/16 network).

```
Authorizer: RAS_ADMIN_KEY_A
Licensees: JIK
Local-Constants:
          RAS_ADMIN_KEY_A = "rsa-base64:MDgCMQDMiEBn89VCSR3ajxr0bNRC\
                 Audlz5724fUaW0uyi4r1oSq8PaSC2v9QGS+phGEahJ8CAwEAAQ=="
          JIK = "x509-base64:MIICGDCCAYGgAwIBAgIBADANBgkqhkiG9w0BAQQ\
                 FADBSMQswCQYDVQQGEwJHQjEOMAwGA1UEChMFQmVuQ28xETAPBg\
                 NVBAMTCEJlbkNvIENBMSAwHgYJKoZIhvcNAQkBFhFiZW5AYWxnc\
                 m91cC5jby51azAeFw050TEwMTEyMzA2MjJaFw050TExMTAyMzA2\
                 MjJaMFIxCzAJBgNVBAYTAkdCMQ4wDAYDVQQKEwVCZW5DbzERMA8\
                 GA1UEAxMIQmVuQ28gQ0ExIDAeBgkqhkiG9w0BCQEWEWJlbkBhbG\
                 dyb3VwLmNvLnVrMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg\
                 QDaCs+JAB6YRKAVkoi1NkOpE1V3syApjBj0Ahjq5HqYAACo1JhM\
                 +QsPwuSWCNhBT51HX6G6UzfY3mOUz/vou6MJ/wor8EdeTX4nucx\
                 NSz/r6XI262aXezAp+GdBviuJZx3Q67ON/IWYrB4QtvihI4bMn5\
                 E55nF6TKtUMJTdATvs/wIDAQABMA0GCSqGSIb3DQEBBAUAA4GBA\
                 MaQ0SkaiR8id0h6Zo0VSB4HpBnjpWqz1jNG8N4RPN0W8muRA2b9\
                 85GNP1bkC3fK1ZPpFTB0A76lLn11CfhAf/gV1iz3ELlUHo5J8nx\
                 Pu6XfsGJm3HsXJ0uv0og8Aean40Do4KInuAsnbLzpGl0d+Jqa5u\
                 TZUxsyg4Q0BwYEU92H"
Conditions: app_domain == "IPsec policy" && doi == "ipsec"
         && pfs == "yes"
         && esp_present == "yes" && ah_present == "no"
         && ( ( esp_enc_alg == "des" && esp_auth_alg == "hmac-md5"
         && remote_filter_proto == "tcp"
         && local_filter_proto == "tcp"
         && ( remote_filter_port == "25"
             || remote_filter_port == "110" ) )
           || ( esp_enc_alg == "3des" && esp_aut_alg == "hmac-sha" ) ) ;
Signature: "sig-rsa-sha1-base64:KhKUeJ6m1zF7kehwHb7W0xAQ8EkPNKbUqNhf/i+f\
            ymBqjbzMy13OmH1itijbFLQJ"
```

Fig. 8. Mobile host credential.