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
University of Pennsylvania
angelos@cis.upenn.edu

Abstract

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.

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 over a particular set of links), edge-to-edge (protecting traffic as it passes between "trusted" networks via an "untrusted" one, subsuming many of the current functions performed by network firewalls), 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 be 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 in-

[*]This work was supported by DARPA under Contract F39502-99-1-0512-MOD P0001.

troduce 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[12]. 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[11].

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.*[17]

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 [11] makes use of the Subject Alternate Name field in X.509 [8] 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.

[20] 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.

[9] describes a language for specifying communication security policies, heavily oriented toward IPsec and IKE. SPSL is based on the Routing Policy Specification Language (RPSL) [1]. 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.

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 [5] 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 [6] or similar protocols. It is not clear what its applicability to IPsec security policy would be.

RADIUS [19] and its proposed successor, DIAMETER [7], 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 [4].

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 would be based on a public-key or Needham-Schroeder [18] 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 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.

- 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 [3] for checking compliance with security policy provides exactly the interface and abstractions required here.

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 [3]. 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:  "Borris Yeltsin"
Conditions:
 EmailAddress == "yeltsin@kremvax.ru"
```

means that the "POLICY" principal authorizes the "Borris 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 [2].

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.

The first problem is credential discovery and acquisition. 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.

Our solution is straightforward: the host that intends to initiate an IKE exchange can 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 may also provide pointers to other servers where the initiator may find relevant credentials; in fact, the responder may 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 may have its own policy concerning which hosts are allowed to query for credentials.

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.

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). 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 [16, 10]. 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.

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.

[1]The SPD is part of all IPsec implementations[15], 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.

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 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 4. 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 [14] protocol.

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

Using the `passphrase:` tag requires policies to be kept private. To avoid this, a hashed version of the passphrase may be used instead (using for example the `passphrase-sha1-hex:` prefix). In the previous example, this would be `passphrase-sha1-hex:8843d7f92416211de9ebb963ff4ce2812-5932878`).

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 3.4.

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. 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.

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 credentail 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

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

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

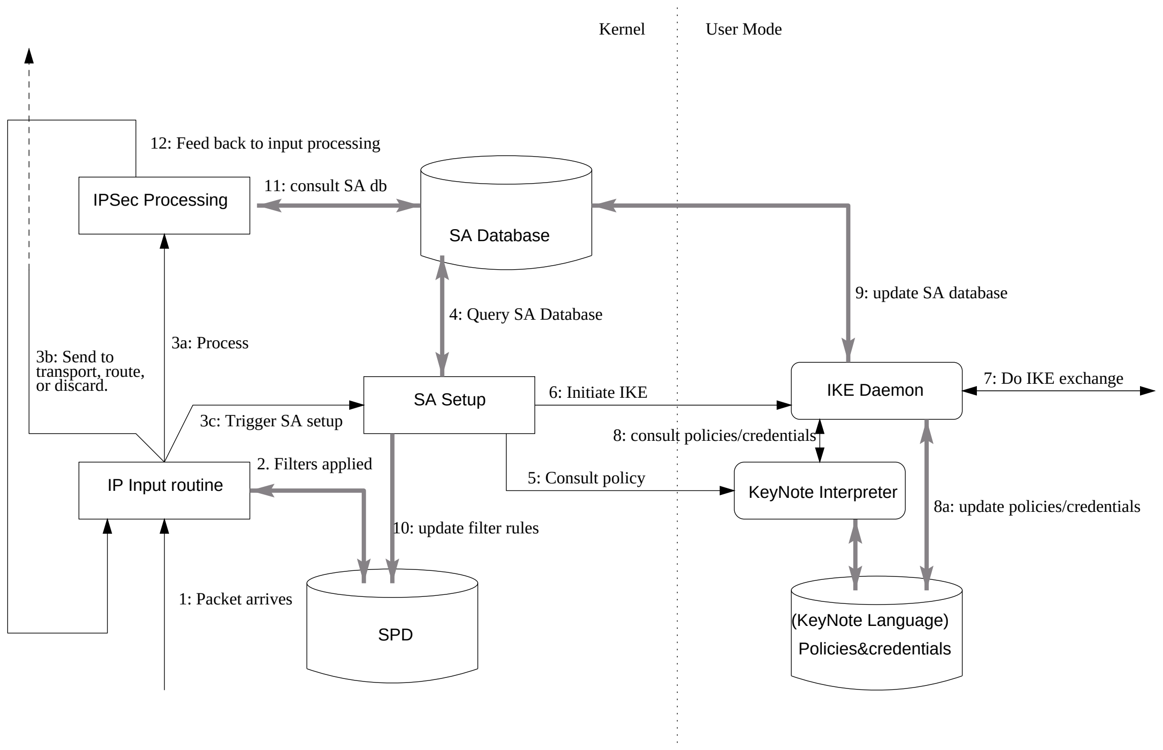

Figure 2. KeyNote-Controlled IPsec Input Processing

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 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.

A host's local policy is given in a text file (`/etc/isakmpd.policy`) that contains KeyNote policy assertions.

Output processing starts when a packet arrives (#1 in Figure 2) 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

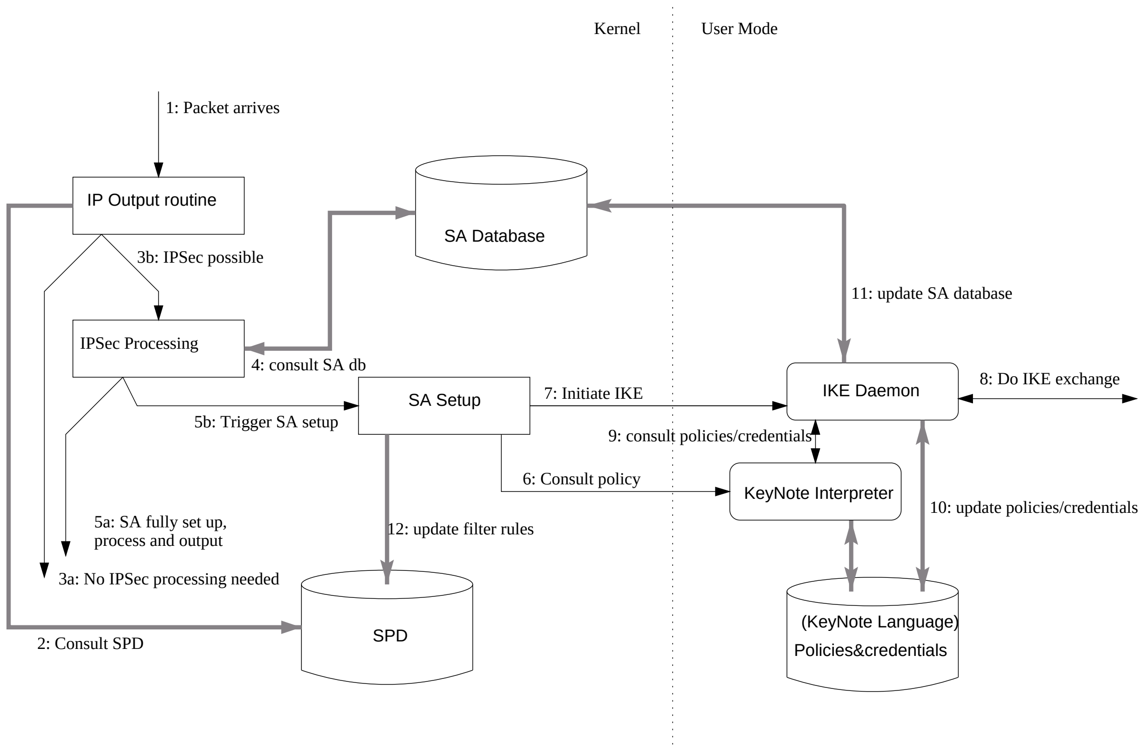

Figure 3. KeyNote-Controlled IPsec Output Processing

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 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 userland 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.conf`. When existing IPsec SAs expire and are subsequently renegotiated, or when new IPsec SAs are established, the new policy will automatically be taken into consideration. If we want to 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 a modern Pentium processor. 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 had to add 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. 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 [21]). 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 [13]. This approach is the subject of ongoing research.

Finally, if trust-management policies and credentials are built into the network security infrastructure, it may be possible to use them as an "intermediate language" between the lower-level protocol and application policy languages (*e.g.,* packet-filtering rules) and higher-level policy specification languages and tools. A translation tool might convert a high-level specification to the trust-management system's language (and perhaps vice-versa as well). Such a tool could make use of formal methods to verify or enforce that the generated policy has certain properties. This approach is currently under investigation in the STRONGMAN DARPA project at the University of Pennsylvania and AT&T Labs.

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).

References

[1] 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.

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

[3] 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.

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

[5] 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.

[6] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. Internet RFC 2208, 1997.

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

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

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

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

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

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

[13] S. Ioannidis, A. Keromytis, S. Bellovin, and J. Smith. Implementing a Distributed Firewall. In *Proceedings of Computer and Communications Security (CCS) 2000*, November 2000.

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

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

[16] A. D. Keromytis, J. Ioannidis, and J. M. Smith. Implementing IPsec. In *Proceedings of Global Internet (GlobeCom) '97*, pages 1948 – 1952, November 1997.

[17] S. McCanne and V. 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, Jan. 1993. Usenix.

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

[19] 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, Apr. 1997.

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

[21] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Protocol Architecture. Internet Draft, Internet Engineering Task Force, Feb. 1999. Work in progress.

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).

```
Authorizer: "POLICY"
Licensees: "passphrase:pedomellonamino"
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" ;
```

Figure 4. Policy for Firewall of 135.207.0.0/16 Network.

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 4 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.

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 4 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 4; 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: 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 ;
```

Figure 5. Mobile host local policy.

```
Authorizer: RAS_ADMIN_KEY_A
Licensees: JIK
Local-Constants:
          RAS_ADMIN_KEY_A = "rsa-base64:MDgCMQDMiEBn89VCSR3ajxr0bNRC\
                 Audlz5724fUaW0uyi4r1oSq8PaSC2v9QGS+phGEahJ8CAwEAAQ=="
          JIK = "x509-base64:MIICGDCCAYGgAwIBAgIBADANBgkqhkiG9w0BAQQ\
                 FADBSMQswCQYDVQQGEwJHQjEOMAwGA1UEChMFQmVuQ28xETAPBg\
                 NVBAMTCEJlbkNvIENBMSAwHgYJKoZIhvcNAQkBFhFiZW5AYWxnc\
                 m91cC5jby51azAeFw05OTEwMTEyMzA2MjJaFw05OTExMTAyMzA2\
                 MjJaMFIxCzAJBgNVBAYTAkdCMQ4wDAYDVQQKEwVCZW5DbzERMA8\
                 GA1UEAxMIQmVuQ28gQ0ExIDAeBgkqhkiG9w0BCQEWEWJlbkBhbG\
                 dyb3VwLmNvLnVrMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBg\
                 QDaCs+JAB6YRKAVkoi1NkOpE1V3syApjBj0Ahjq5HqYAACo1JhM\
                 +QsPwuSWCNhBT51HX6G6UzfY3mOUz/vou6MJ/wor8EdeTX4nucx\
                 NSz/r6XI262aXezAp+GdBviuJZx3Q67ON/IWYrB4QtvihI4bMn5\
                 E55nF6TKtUMJTdATvs/wIDAQABMA0GCSqGSIb3DQEBBAUAA4GBA\
                 MaQOSkaiR8id0h6Zo0VSB4HpBnjpWqz1jNG8N4RPN0W8muRA2b9\
                 85GNP1bkC3fK1ZPpFTB0A76lLn11CfhAf/gV1iz3ELlUHo5J8nx\
                 Pu6XfsGJm3HsXJOuvOog8Aean4ODo4KInuAsnbLzpGl0d+Jqa5u\
                 TZUxsyg4QOBwYEU92H"
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"
```

Figure 6. Mobile host credential.