S3-000766

IPSP

3GPP SA3 #16

28-30 November 2000

John Ioannidis

AT&T

ji@research.att.com

Policy for IPsec

- AH and ESP provide ***mechanism***.
- IKE does ***key agreement***.

- IPSP manages ***policy***:
  - "How do I use IPsec to talk to another host?"
  - "Is it possible to create an SA that conforms to my policy?"
  - "What should the SA parameters be?"
  - "Who is my security gateway?"
  - "Where do I find my policies?"

Design Space

- Runs on anything that runs IPsec.
- Decentralized and heterogeneous administration.
  - Two nodes need not trust common admin.
  - Remote administration.
  - Delegation.
- SA parameters not specified in advance.
- Secure, sound, comprehensible.
  - Human-readable policies.
  - Compatible with security proofs.
  - Correct implementation should be straighforward.

Requirements

- Policy model.
- IPsec gateway discovery mechanism.
- Policy language for nodes.
- Means of distributing responsibility.
- Protocol for policy discovery.
- Method for resolving SA parameters.
- Compliance checking.
- No changes to AH/ESP/IKE.

Policy Model

- Defines the semantics of IPsec policy.
- Everything (gateway discovery, SA resolution, compliance checking) implements these semantics.
- Independent of specific details (of language, distribution protocols, etc.).

Gateway Discovery

- How a node finds where to direct IPsec traffic for another node.

IPSP Language

- Standard language for representing a node's policy *externally* to other nodes.
  - May be different from local policy configuration mechanisms.
- Output of policy discovery protocol.
- Input to SA resolution and compliance checking steps.

Distributed Policy

- Must be possible to have remote administration of a node's policy.
- Must be possible to delegate authorization and responsibility.
- Must have support for security gateways, remote services, etc.

Policy Discovery

- Protocol that provides information (in IPSP language) about a node's policy to other hosts.
- Node need not reveal its entire policy.
- Just enough to allow others to do SA parameter resolution.

SA Parameter Resolution

- Given output of policy discovery protocol:
  - can two nodes communicate at all?
  - What set of SA parameters meets both nodes' policies?
- Must be computationally efficient to be practical.

Compliance Checking

- Given a set of proposed SA parameters, a node must be able to verify:
  - Whether parameters meet its own policy.
  - Whether gateway is correct.
- This is where policy enforcement is implemented.

Security Policy Protocol

- Protocol for discovering SEGs, distributing policies.
- Generic and extensible.
- Initiator sends message to remote end-host.
  - SEGs intercept and forward to policy server.
  - Policies acquired and forwarded to end-host.
- SEGs can examine acquired policies, changes reqs.
  - Avoid redundant IKE operations.
  - Main reason for bundling discovery and distribution in the same protocol.
- Can be initiated by end-host or firewall.
- Policy Server may be local to a host/SEG.
- Policy Server must be configured.

KeyNote and Compliance Checking

- Standard format for policy distribution and compliance checking.
- Simple, extensible language (RFC 2704).
- Used for expressing policies.
  - SPD/SA parameters.
  - Trusted peers/third parties.
  - Integrity-protected.
- Allows authorization delegation.
  - Various types of trust relations between security domains.

KeyNote

- A Trust-Management System.
- Compliance Checking:
  - determines if Actions are compatible with Policies.
- Human-readable policies.
- Wide variety of applications:
  - IPsec policy
  - Workflow.
  - Digital Rights Management System
  - Micropayments System
  - Kernel policy management.

Actions

- ***Actions*** are activities that have security considerations.
- In KeyNote, actions are described by a set of attribute-value pairs called the **Action Environment**.
- Attribute semantics depend on the application
- An Action is always associated with a Requestor.

  Requestor may be a public key, a user name, etc.

Policies

- ***Policies*** determine *who* is trusted to authorize various actions.
- In KeyNote, Policies are a collection of Assertions.
- Assertions determine whether a Requestor is authorized to request an Action.
- Two major components to Assertions:
  - **Licencees**: checks who the requestors may be.
  - **Conditions**: checks the Action Environment.
- Licencees and Conditions are programmable expressions.
- Other components provide additional semantic structure (comments, identification, *etc*.).

Sample Assertion

```
Authorizer:    POLICY
Licencees:     wendy
Conditions:    $file_owner == “stan”
                  && $filename ~= “/home/stan/[^/]*”
                  ->  { return TRUE }
```

Turning policies into credentials

- We have shown how assertions authorize requestors.
- Assertions may also defer to other assertions.
- An assertion may be signed and used as a cryptographic credential.

Authorizer: *stan's public key*
Licencees: *wendy's public key*
Conditions: $file_owner == "stan"
&& $filename ~= "/home/stan/[^/]*"
-> { return TRUE }
Signature: …..

Evaluation of a Request

- KeyNote is a compliance checker.
- Determines whether requested action satisfies policy.
- Finds a subgraph of assertions linking action to POLICY.
- For precise semantics, see the draft.

IPsec example

- TM for IPsec paper

Details

- The **Licencees** field may contain:
  - single identifier.
  - A complex expression.
- Expressions are:
  - monotonic (important for security proofs).
  - Disjunction, conjunction, threshold.

Observations

- **Conditions** may return more than just TRUE/FALSE.
- It may also pass back information to the application.
- It can work in conjunction with X.509 and SDSI names.

Pointers

- IP Security Policy IETF Working Group:

http://www.ietf.org/html.charters/ipsp-charter.html

- Trust Management:

http://www.crypto.com/trustmgt

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

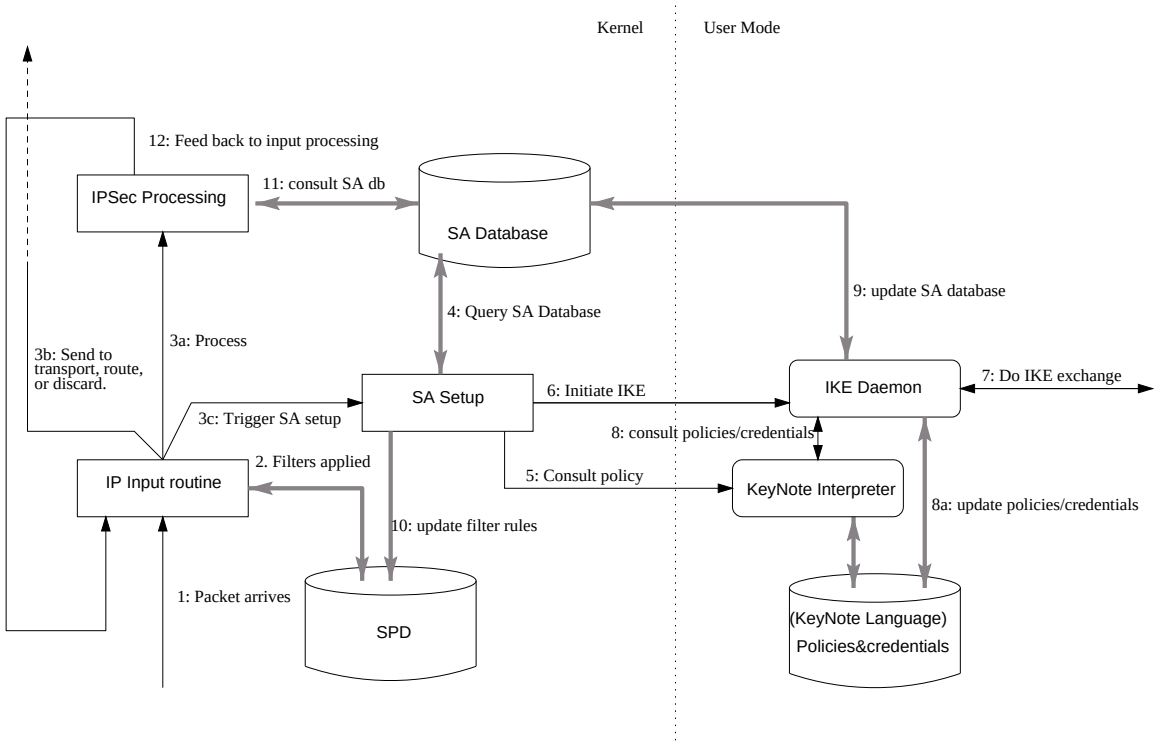

Figure 2. KeyNote-Controlled IPsec Input Processing

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

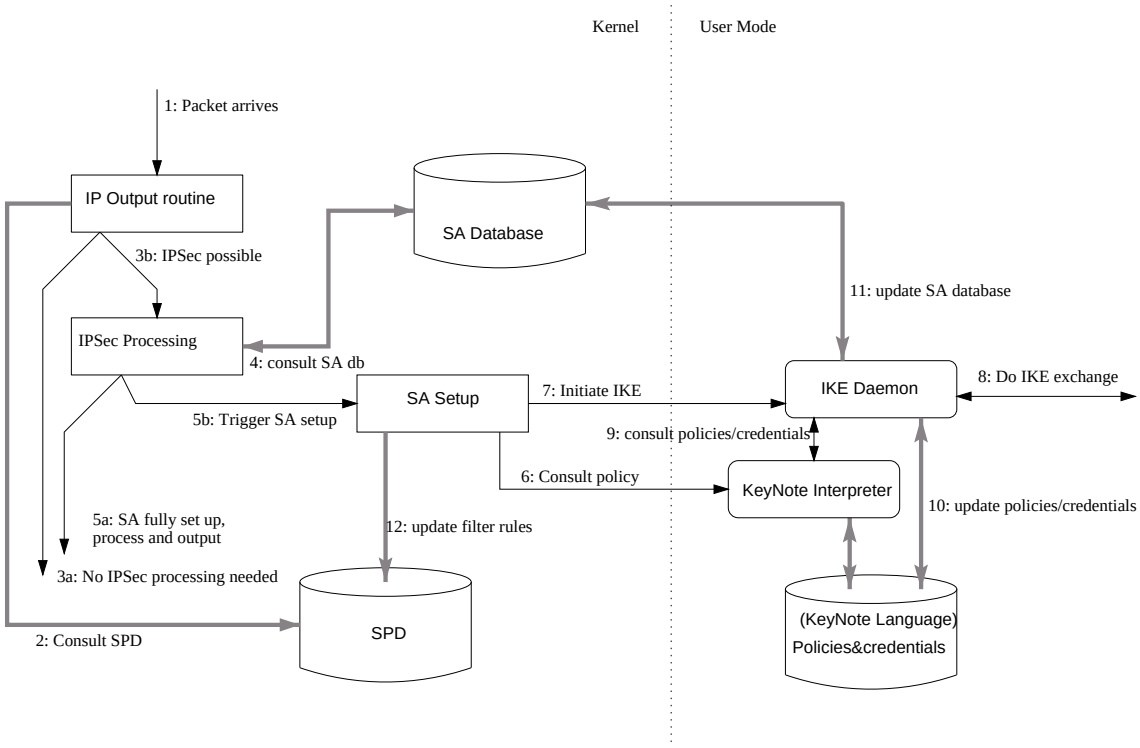

Figure 3. KeyNote-Controlled IPsec Output Processing

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:

```
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.

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 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.

Network Working Group
Request for Comments: 2704
Category: Informational

M. Blaze
J. Feigenbaum
J. Ioannidis
AT&T Labs - Research
A. Keromytis
U. of Pennsylvania

September 1999

The KeyNote Trust-Management System Version 2

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This memo describes version 2 of the KeyNote trust-management system. It specifies the syntax and semantics of KeyNote 'assertions', describes 'action attribute' processing, and outlines the application architecture into which a KeyNote implementation can be fit. The KeyNote architecture and language are useful as building blocks for the trust management aspects of a variety of Internet protocols and services.

1. Introduction

Trust management, introduced in the PolicyMaker system [BFL96], is a unified approach to specifying and interpreting security policies, credentials, and relationships; it allows direct authorization of security-critical actions. A trust-management system provides standard, general-purpose mechanisms for specifying application security policies and credentials. Trust-management credentials describe a specific delegation of trust and subsume the role of public key certificates; unlike traditional certificates, which bind keys to names, credentials can bind keys directly to the authorization to perform specific tasks.

A trust-management system has five basic components:

* A language for describing 'actions', which are operations with security consequences that are to be controlled by the system.

* A mechanism for identifying 'principals', which are entities that can be authorized to perform actions.

* A language for specifying application 'policies', which govern the actions that principals are authorized to perform.

* A language for specifying 'credentials', which allow principals to delegate authorization to other principals.

* A 'compliance checker', which provides a service to applications for determining how an action requested by principals should be handled, given a policy and a set of credentials.

The trust-management approach has a number of advantages over other mechanisms for specifying and controlling authorization, especially when security policy is distributed over a network or is otherwise decentralized.

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

Trust-management policies are easy to distribute across networks, helping to avoid the need for application-specific distributed policy configuration mechanisms, access control lists, and certificate parsers and interpreters.

For a general discussion of the use of trust management in distributed system security, see [Bla99].

KeyNote is a simple and flexible trust-management system designed to work well for a variety of large- and small-scale Internet-based applications. It provides a single, unified language for both local policies and credentials. KeyNote policies and credentials, called 'assertions', contain predicates that describe the trusted actions permitted by the holders of specific public keys. KeyNote assertions are essentially small, highly-structured programs. A signed

assertion, which can be sent over an untrusted network, is also called a 'credential assertion'. Credential assertions, which also serve the role of certificates, have the same syntax as policy assertions but are also signed by the principal delegating the trust.

In KeyNote:

* Actions are specified as a collection of name-value pairs.

* Principal names can be any convenient string and can directly represent cryptographic public keys.

* The same language is used for both policies and credentials.

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

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

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

This document describes the KeyNote policy and credential assertion language, the structure of KeyNote action descriptions, and the KeyNote model of computation.

We assume that applications communicate with a locally trusted KeyNote compliance checker via a 'function call' style interface, sending a collection of KeyNote policy and credential assertions plus an action description as input and accepting the resulting policy compliance value as output. However, the requirements of different applications, hosts, and environments may give rise to a variety of different interfaces to KeyNote compliance checkers; this document does not aim to specify a complete compliance checker API.

2. KeyNote Concepts

In KeyNote, the authority to perform trusted actions is associated with one or more 'principals'. A principal may be a physical entity, a process in an operating system, a public key, or any other convenient abstraction. KeyNote principals are identified by a string called a 'Principal Identifier'. In some cases, a Principal Identifier will contain a cryptographic key interpreted by the

KeyNote system (e.g., for credential signature verification). In other cases, Principal Identifiers may have a structure that is opaque to KeyNote.

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

Actions are described to the KeyNote compliance checker in terms of a collection of name-value pairs called an ‘action attribute set’. The action attribute set is created by the invoking application. Its structure and format are described in detail in Section 3 of this document.

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

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

Assertions are the basic programming unit for specifying policy and delegating authority. Assertions describe the conditions under which a principal authorizes actions requested by other principals. An assertion identifies the principal that made it, which other principals are being authorized, and the conditions under which the authorization applies. The syntax of assertions is given in Section 4.

A special principal, whose identifier is "POLICY", provides the root of trust in KeyNote. "POLICY" is therefore considered to be authorized to perform any action.

Assertions issued by the "POLICY" principal are called ‘policy assertions’ and are used to delegate authority to otherwise untrusted principals. The KeyNote security policy of an application consists of a collection of policy assertions.

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

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

```
KeyNote-Version: 2
Comment: A simple, if contrived, email certificate for user mab
Local-Constants:  ATT_CA_key = "RSA:acdfa1df1011bbac"
                  mab_key = "DSA:deadbeefcafe001a"
Authorizer: ATT_CA_key
Licensees: mab_key
Conditions: ((app_domain == "email")  # valid for email only
          && (address == "mab@research.att.com"));
Signature: "RSA-SHA1:f00f2244"
```

The meanings of the various sections are described in Sections 4 and 5 of this document.

KeyNote semantics resolve the relationship between an application’s policy and actions requested by other principals, as supported by credentials. The KeyNote compliance checker processes the assertions against the action attribute set to determine the policy compliance value of a requested action. These semantics are defined in Section 5.

An important principle in KeyNote’s design is ‘assertion monotonicity’; the policy compliance value of an action is always positively derived from assertions made by trusted principals. Removing an assertion never results in increasing the compliance value returned by KeyNote for a given query. The monotonicity

property can simplify the design and analysis of complex network-based security protocols; network failures that prevent the transmission of credentials can never result in spurious authorization of dangerous actions. A detailed discussion of monotonicity and safety in trust management can be found in [BFL96] and [BFS98].

3. Action Attributes

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

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

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

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

Attribute names are of the form:

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

That is, an <AttributeID> begins with an alphabetic or underscore character and can be followed by any number of alphanumerics and underscores. Attribute names are case-sensitive.

The exact mechanism for passing the action attribute set to the compliance checker is determined by the KeyNote implementation. Depending on specific requirements, an implementation may provide a mechanism for including the entire attribute set as an explicit parameter of the query, or it may provide some form of callback mechanism invoked as each attribute is dereferenced, e.g., for access to kernel variables.

If an action attribute is not defined its value is considered to be the empty string.

Attribute names beginning with the "_" character are reserved for use by the KeyNote runtime environment and cannot be passed from applications as part of queries. The following special attribute names are used:

Name	Purpose
_MIN_TRUST	Lowest-order (minimum) compliance value in query; see Section 5.1.
_MAX_TRUST	Highest-order (maximum) compliance value in query; see Section 5.1.
_VALUES	Linearly ordered set of compliance values in query; see Section 5.1. Comma separated.
_ACTION_AUTHORIZERS	Names of principals directly authorizing action in query. Comma separated.

In addition, attributes with names of the form "_<N>", where <N> is an ASCII-encoded integer, are used by the regular expression matching mechanism described in Section 5.

The assignment and semantics of any other attribute names beginning with "_" is unspecified and implementation-dependent.

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

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

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

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

Note that "RFC822-EMAIL" is a hypothetical example; such a name may or may not appear in the actual registry with these or different attributes. (Indeed, we recognize that the reality of email security is considerably more complex than this example might suggest.)

4. KeyNote Assertion Syntax

In the following sections, the notation [X]* means zero or more repetitions of character string X. The notation [X]+ means one or more repetitions of X. The notation <X>* means zero or more repetitions of non-terminal <X>. The notation <X>+ means one or more repetitions of X, whereas <X>? means zero or one repetitions of X. Nonterminal grammar symbols are enclosed in angle brackets. Quoted strings in grammar productions represent terminals.

4.1 Basic Structure

```
<Assertion>:: <VersionField>? <AuthField> <LicenseesField>?
              <LocalConstantsField>? <ConditionsField>?
              <CommentField>? <SignatureField>? ;
```

All KeyNote assertions are encoded in ASCII.

KeyNote assertions are divided into sections, called ‘fields’, that serve various semantic functions. Each field starts with an identifying label at the beginning of a line, followed by the ":" character and the field’s contents. There can be at most one field per line.

A field may be continued over more than one line by indenting subsequent lines with at least one ASCII SPACE or TAB character. Whitespace (a SPACE, TAB, or NEWLINE character) separates tokens but is otherwise ignored outside of quoted strings. Comments with a leading octothorp character (see Section 4.2) may begin in any column.

One mandatory field is required in all assertions:

Authorizer

Six optional fields may also appear:

Comment
Conditions
KeyNote-Version
Licensees
Local-Constants
Signature

All field names are case-insensitive. The "KeyNote-Version" field, if present, appears first. The "Signature" field, if present, appears last. Otherwise, fields may appear in any order. Each field may appear at most once in any assertion.

Blank lines are not permitted in assertions. Multiple assertions stored in a file (e.g., in application policy configurations), therefore, can be separated from one another unambiguously by the use of blank lines between them.

4.2 Comments

<Comment>:: "#" {ASCII characters} ;

The octothorp character ("#", ASCII 35 decimal) can be used to introduce comments. Outside of quoted strings (see Section 4.3), all characters from the "#" character through the end of the current line are ignored. However, commented text is included in the computation of assertion signatures (see Section 4.6.7).

4.3 Strings

A ‘string’ is a lexical object containing a sequence of characters. Strings may contain any non-NUL characters, including newlines and nonprintable characters. Strings may be given as literals, computed from complex expressions, or dereferenced from attribute names.

4.3.1 String Literals

```
<StringLiteral>:: "\"" {see description below} "\"" ;
```

A string literal directly represents the value of a string. String literals must be quoted by preceding and following them with the double-quote character (ASCII 34 decimal).

A printable character may be ‘escaped’ inside a quoted string literal by preceding it with the backslash character (ASCII 92 decimal) (e.g., "like \"this\"."). This permits the inclusion of the double-quote and backslash characters inside string literals.

A similar escape mechanism is also used to represent non-printable characters. "\n" represents the newline character (ASCII character 10 decimal), "\r" represents the carriage-return character (ASCII character 13 decimal), "\t" represents the tab character (ASCII character 9 decimal), and "\f" represents the form-feed character (ASCII character 12 decimal). A backslash character followed by a newline suppresses all subsequent whitespace (including the newline) up to the next non-whitespace character (this allows the continuation of long string constants across lines). Un-escaped newline and return characters are illegal inside string literals.

The constructs "\0o", "\0oo", and "\ooo" (where o represents any octal digit) may be used to represent any non-NUL ASCII characters with their corresponding octal values (thus, "\012" is the same as "\n", "\101" is "A", and "\377" is the ASCII character 255 decimal). However, the NUL character cannot be encoded in this manner; "\0", "\00", and "\000" are converted to the strings "0", "00", and "000" respectively. Similarly, all other escaped characters have the leading backslash removed (e.g., "\a" becomes "a", and "\\" becomes "\"). The following four strings are equivalent:

```
"this string contains a newline\n followed by one space."
"this string contains a newline\n \
followed by one space."
```

```
"this str\
   ing contains a \
     newline\n followed by one space."

"this string contains a newline\012\040followed by one space."
```

4.3.2 String Expressions

In general, anywhere a quoted string literal is allowed, a ‘string expression’ can be used. A string expression constructs a string from string constants, dereferenced attributes (described in Section 4.4), and a string concatenation operator. String expressions may be parenthesized.

```
<StrEx>:: <StrEx> "." <StrEx>    /* String concatenation */
        | <StringLiteral>         /* Quoted string */
        | "(" <StrEx> ")"
        | <DerefAttribute>        /* See Section 4.4 */
        | "$" <StrEx> ;           /* See Section 4.4 */
```

The "$" operator has higher precedence than the "." operator.

4.4 Dereferenced Attributes

Action attributes provide the primary mechanism for applications to pass information to assertions. Attribute names are strings from a limited character set (<AttributeID> as defined in Section 3), and attribute values are represented internally as strings. An attribute is dereferenced simply by using its name. In general, KeyNote allows the use of an attribute anywhere a string literal is permitted.

Attributes are dereferenced as strings by default. When required, dereferenced attributes can be converted to integers or floating point numbers with the type conversion operators "@" and "&". Thus, an attribute named "foo" having the value "1.2" may be interpreted as the string "1.2" (foo), the integer value 1 (@foo), or the floating point value 1.2 (&foo).

Attributes converted to integer and floating point numbers are represented according to the ANSI C ‘long’ and ‘float’ types, respectively. In particular, integers range from -2147483648 to 2147483647, whilst floats range from 1.17549435E-38F to 3.40282347E+38F.

Any uninitialized attribute has the empty-string value when dereferenced as a string and the value zero when dereferenced as an integer or float.

Attribute names may be given literally or calculated from string expressions and may be recursively dereferenced. In the simplest case, an attribute is dereferenced simply by using its name outside of quotes; e.g., the string value of the attribute named "foo" is by reference to ‘foo’ (outside of quotes). The "$<StrEx>" construct dereferences the attribute named in the string expression <StrEx>. For example, if the attribute named "foo" contains the string "bar", the attribute named "bar" contains the string "xyz", and the attribute "xyz" contains the string "qua", the following string comparisons are all true:

```
foo == "bar"
$("foo") == "bar"
$foo == "xyz"
$(foo) == "xyz"
$$foo == "qua"
```

If <StrEx> evaluates to an invalid or uninitialized attribute name, its value is considered to be the empty string (or zero if used as a numeric).

The <DerefAttribute> token is defined as:

```
   <DerefAttribute>:: <AttributeID> ;
```

4.5 Principal Identifiers

Principals are represented as ASCII strings called ‘Principal Identifiers’. Principal Identifiers may be arbitrary labels whose structure is not interpreted by the KeyNote system or they may encode cryptographic keys that are used by KeyNote for credential signature verification.

```
    <PrincipalIdentifier>:: <OpaqueID>
                          | <KeyID> ;
```

4.5.1 Opaque Principal Identifiers

Principal Identifiers that are used by KeyNote only as labels are said to be ‘opaque’. Opaque identifiers are encoded in assertions as strings (see Section 4.3):

```
    <OpaqueID>:: <StrEx> ;
```

Opaque identifier strings should not contain the ":" character.

4.5.2 Cryptographic Principal Identifiers

Principal Identifiers that are used by KeyNote as keys, e.g., to verify credential signatures, are said to be ‘cryptographic’. Cryptographic identifiers are also lexically encoded as strings:

```
<KeyID>:: <StrEx> ;
```

Unlike Opaque Identifiers, however, Cryptographic Identifier strings have a special form. To be interpreted by KeyNote (for signature verification), an identifier string should be of the form:

```
<IDString>:: <ALGORITHM>":"<ENCODEDBITS> ;
```

"ALGORITHM" is an ASCII substring that describes the algorithms to be used in interpreting the key’s bits. The ALGORITHM identifies the major cryptographic algorithm (e.g., RSA [RSA78], DSA [DSA94], etc.), structured format (e.g., PKCS1 [PKCS1]), and key bit encoding (e.g., HEX or BASE64). By convention, the ALGORITHM substring starts with an alphabetic character and can contain letters, digits, underscores, or dashes (i.e., it should match the regular expression "[a-zA-Z][a-zA-Z0-9_-]*"). The IANA (or some other appropriate authority) will provide a registry of reserved algorithm identifiers.

"ENCODEDBITS" is a substring of characters representing the key’s bits, the encoding and format of which depends on the ALGORITHM. By convention, hexadecimal encoded keys use lower-case ASCII characters.

Cryptographic Principal Identifiers are converted to a normalized canonical form for the purposes of any internal comparisons between them; see Section 5.2.

Note that the keys used in examples throughout this document are fictitious and generally much shorter than would be required for security in practice.

4.6 KeyNote Fields

4.6.1 The KeyNote-Version Field

The KeyNote-Version field identifies the version of the KeyNote assertion language under which the assertion was written. The KeyNote-Version field is of the form

```
<VersionField>:: "KeyNote-Version:" <VersionString> ;
<VersionString>:: <StringLiteral>
                | <IntegerLiteral> ;
```

where <VersionString> is an ASCII-encoded string. Assertions in production versions of KeyNote use decimal digits in the version representing the version number of the KeyNote language under which they are to be interpreted. Assertions written to conform with this document should be identified with the version string "2" (or the integer 2). The KeyNote-Version field, if included, should appear first.

4.6.2 The Local-Constants Field

This field adds or overrides action attributes in the current assertion only. This mechanism allows the use of short names for (frequently lengthy) cryptographic principal identifiers, especially to make the Licensees field more readable. The Local-Constants field is of the form:

```
<LocalConstantsField>:: "Local-Constants:" <Assignments> ;
<Assignments>:: /* can be empty */
              | <AttributeID> "=" <StringLiteral> <Assignments> ;
```

<AttributeID> is an attribute name from the action attribute namespace as defined in Section 3. The name is available for use as an attribute in any subsequent field. If the Local-Constants field defines more than one identifier, it can occupy more than one line and be indented. <StringLiteral> is a string literal as described in Section 4.3. Attributes defined in the Local-Constants field override any attributes with the same name passed in with the action attribute set.

An attribute may be initialized at most once in the Local-Constants field. If an attribute is initialized more than once in an assertion, the entire assertion is considered invalid and is not considered by the KeyNote compliance checker in evaluating queries.

4.6.3 The Authorizer Field

The Authorizer identifies the Principal issuing the assertion. This field is of the form

```
<AuthField>:: "Authorizer:" <AuthID> ;
<AuthID>:: <PrincipalIdentifier>
         | <DerefAttribute> ;
```

The Principal Identifier may be given directly or by reference to the attribute namespace (as defined in Section 4.4).

4.6.4 The Licensees Field

The Licensees field identifies the principals authorized by the assertion. More than one principal can be authorized, and authorization can be distributed across several principals through the use of ‘and’ and threshold constructs. This field is of the form

```
<LicenseesField>:: "Licensees:" <LicenseesExpr> ;

<LicenseesExpr>::      /* can be empty */
                  | <PrincExpr> ;

<PrincExpr>:: "(" <PrincExpr> ")"
              | <PrincExpr> "&&" <PrincExpr>
              | <PrincExpr> "||" <PrincExpr>
              | <K>"-of(" <PrincList> ")"        /* Threshold */
              | <PrincipalIdentifier>
              | <DerefAttribute> ;

<PrincList>:: <PrincipalIdentifier>
            | <DerefAttribute>
            | <PrincList> "," <PrincList> ;

<K>:: {Decimal number starting with a digit from 1 to 9} ;
```

The "&&" operator has higher precedence than the "||" operator. <K> is an ASCII-encoded positive decimal integer. If a <PrincList> contains fewer than <K> principals, the entire assertion is omitted from processing.

4.6.5 The Conditions Field

This field gives the ‘conditions’ under which the Authorizer trusts the Licensees to perform an action. ‘Conditions’ are predicates that operate on the action attribute set. The Conditions field is of the form:

```
<ConditionsField>:: "Conditions:" <ConditionsProgram> ;

<ConditionsProgram>:: /* Can be empty */
                    | <Clause> ";" <ConditionsProgram> ;

<Clause>:: <Test> "->" "{" <ConditionsProgram> "}"
         | <Test> "->" <Value>
         | <Test> ;

<Value>:: <StrEx> ;
```

```
    <Test>:: <RelExpr> ;

    <RelExpr>:: "(" <RelExpr> ")"        /* Parentheses */
              | <RelExpr> "&&" <RelExpr> /* Logical AND */
              | <RelExpr> "||" <RelExpr> /* Logical OR */
              | "!" <RelExpr>         /* Logical NOT */
              | <IntRelExpr>
              | <FloatRelExpr>
              | <StringRelExpr>
              | "true"        /* case insensitive */
              | "false" ;     /* case insensitive */

    <IntRelExpr>:: <IntEx> "==" <IntEx>
                 | <IntEx> "!=" <IntEx>
                 | <IntEx> "<" <IntEx>
                 | <IntEx> ">" <IntEx>
                 | <IntEx> "<=" <IntEx>
                 | <IntEx> ">=" <IntEx> ;

    <FloatRelExpr>:: <FloatEx> "<" <FloatEx>
                   | <FloatEx> ">" <FloatEx>
                   | <FloatEx> "<=" <FloatEx>
                   | <FloatEx> ">=" <FloatEx> ;

    <StringRelExpr>:: <StrEx> "==" <StrEx>  /* String equality */
                    | <StrEx> "!=" <StrEx>  /* String inequality */
                    | <StrEx> "<" <StrEx>   /* Alphanum. comparisons */
                    | <StrEx> ">" <StrEx>
                    | <StrEx> "<=" <StrEx>
                    | <StrEx> ">=" <StrEx>
                    | <StrEx> "~=" <RegExpr> ; /* Reg. expr. matching */

    <IntEx>:: <IntEx> "+" <IntEx>        /* Integer */
            | <IntEx> "-" <IntEx>
            | <IntEx> "*" <IntEx>
            | <IntEx> "/" <IntEx>
            | <IntEx> "%" <IntEx>
            | <IntEx> "^" <IntEx>        /* Exponentiation */
            | "-" <IntEx>
            | "(" <IntEx> ")"
            | <IntegerLiteral>
            | "@" <StrEx> ;

    <FloatEx>:: <FloatEx> "+" <FloatEx>  /* Floating point */
              | <FloatEx> "-" <FloatEx>
              | <FloatEx> "*" <FloatEx>
              | <FloatEx> "/" <FloatEx>
              | <FloatEx> "^" <FloatEx> /* Exponentiation */
```

```
            | "-" <FloatEx>
            | "(" <FloatEx> ")"
            | <FloatLiteral>
            | "&" <StrEx> ;

 <IntegerLiteral>:: {Decimal number of at least one digit} ;
 <FloatLiteral>:: <IntegerLiteral>"."<IntegerLiteral> ;
```

<StringLiteral> is a quoted string as defined in Section 4.3
<AttributeID> is defined in Section 3.

The operation precedence classes are (from highest to lowest):

```
     { (, ) }
     {unary -, @, &, $}
     {^}
     {*, /, %}
     {+, -, .}
```

Operators in the same precedence class are evaluated left-to-right.

Note the inability to test for floating point equality, as most floating point implementations (hardware or otherwise) do not guarantee accurate equality testing.

Also note that integer and floating point expressions can only be used within clauses of condition fields, but in no other KeyNote field.

The keywords "true" and "false" are not reserved; they can be used as attribute or principal identifier names (although this practice makes assertions difficult to understand and is discouraged).

<RegExpr> is a standard regular expression, conforming to the POSIX 1003.2 regular expression syntax and semantics.

Any string expression (or attribute) containing the ASCII representation of a numeric value can be converted to an integer or float with the use of the "@" and "&" operators, respectively. Any fractional component of an attribute value dereferenced as an integer is rounded down. If an attribute dereferenced as a number cannot be properly converted (e.g., it contains invalid characters or is empty) its value is considered to be zero.

4.6.6 The Comment Field

The Comment field allows assertions to be annotated with information describing their purpose. It is of the form

```
<CommentField>:: "Comment:" <text> ;
```

No interpretation of the contents of this field is performed by KeyNote. Note that this is one of two mechanisms for including comments in KeyNote assertions; comments can also be inserted anywhere in an assertion's body by preceding them with the "#" character (except inside string literals).

4.6.7 The Signature Field

The Signature field identifies a signed assertion and gives the encoded digital signature of the principal identified in the Authorizer field. The Signature field is of the form:

```
<SignatureField>:: "Signature:" <Signature> ;

<Signature>:: <StrEx> ;
```

The <Signature> string should be of the form:

```
<IDString>:: <ALGORITHM>":"<ENCODEDBITS> ;
```

The formats of the "ALGORITHM" and "ENCODEDBITS" substrings are as described for Cryptographic Principal Identifiers in Section 4.4.2 The algorithm name should be the same as that of the principal appearing in the Authorizer field. The IANA (or some other suitable authority) will provide a registry of reserved names. It is not necessary that the encodings of the signature and the authorizer key be the same.

If the signature field is included, the principal named in the Authorizer field must be a Cryptographic Principal Identifier, the algorithm must be known to the KeyNote implementation, and the signature must be correct for the assertion body and authorizer key.

The signature is computed over the assertion text, beginning with the first field (including the field identifier string), up to (but not including) the Signature field identifier. The newline preceding the signature field identifier is the last character included in signature calculation. The signature is always the last field in a KeyNote assertion. Text following this field is not considered part of the assertion.

The algorithms for computing and verifying signatures must be configured into each KeyNote implementation and are defined and documented separately.

Note that all signatures used in examples in this document are fictitious and generally much shorter than would be required for security in practice.

5. Query Evaluation Semantics

The KeyNote compliance checker finds and returns the Policy Compliance Value of queries, as defined in Section 5.3, below.

5.1 Query Parameters

A KeyNote query has four parameters:

* The identifier of the principal(s) requesting the action.
* The action attribute set describing the action.
* The set of compliance values of interest to the application, ordered from _MIN_TRUST to _MAX_TRUST
* The policy and credential assertions that should be included in the evaluation.

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

5.1.1 Action Requester

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

5.1.2 Ordered Compliance Value Set

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

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

5.2 Principal Identifier Normalization

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

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

Opaque identifiers are compared as case-sensitive strings.

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

5.3 Policy Compliance Value Calculation

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

5.3.1 Principal Compliance Value

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

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

5.3.2 Direct Authorization Value

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

5.3.3 Assertion Compliance Value

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

5.3.4 Conditions Compliance Value

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

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

If an assertion's Conditions field is missing entirely, its Conditions Compliance Value is considered to be the _MAX_TRUST value, as defined in Section 5.1.

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

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

```
    foo=="bar";
```

is equivalent to

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

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

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

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

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

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

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

is equivalent to

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

String comparisons are case-sensitive.

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

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

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

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

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

5.3.5 Licensee Compliance Value

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

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

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

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

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

* A "&&" expression has the lower-order (minimum) of its two subexpression values.

* A "||" expression has the higher-order (maximum) of its two subexpression values.

* A "<K>-of(<List>)" expression has the K-th highest order compliance value listed in <list>. Values that appear multiple times are counted with multiplicity. For example, if K = 3 and the orders of the listed compliance values are (0, 1, 2, 2, 3), the value of the expression is the compliance value of order 2.

For example, consider the following Licensees field:

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

If the Principal Compliance Value is "yes" for principal "alice", "no" for principal "bob", and "no" for principal "eve", and "yes" is higher order than "no" in the query's Compliance Value Set, then the resulting Licensee Compliance Value is "no".

Observe that if there are exactly two possible compliance values (e.g., "false" and "true"), the rules of Licensee Compliance Value resolution reduce exactly to standard Boolean logic.

5.4 Assertion Management

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

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

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

5.5 Implementation Issues

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

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

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

6. Examples

In this section, we give examples of KeyNote assertions that might be used in hypothetical applications. These examples are intended primarily to illustrate features of KeyNote assertion syntax and semantics, and do not necessarily represent the best way to integrate KeyNote into applications.

In the interest of readability, we use much shorter keys than would ordinarily be used in practice. Note that the Signature fields in these examples do not represent the result of any real signature calculation.

1. TRADITIONAL CA / EMAIL

A. A policy unconditionally authorizing RSA key abc123 for all actions. This essentially defers the ability to specify policy to the holder of the secret key corresponding to abc123:

```
Authorizer: "POLICY"
Licensees: "RSA:abc123"
```

B. A credential assertion in which RSA Key abc123 trusts either RSA key 4401ff92 (called ‘Alice’) or DSA key d1234f (called ‘Bob’) to perform actions in which the "app_domain" is "RFC822-EMAIL", where the "address" matches the regular expression "^.*@keynote\.research\.att\.com$". In other words, abc123 trusts Alice and Bob as certification authorities for the keynote.research.att.com domain.

```
KeyNote-Version: 2
Local-Constants: Alice="DSA:4401ff92"  # Alice’s key
                 Bob="RSA:d1234f"      # Bob’s key
Authorizer: "RSA:abc123"
Licensees: Alice || Bob
Conditions: (app_domain == "RFC822-EMAIL") &&
            (address ~=   # only applies to one domain
              "^.*@keynote\\.research\\.att\\.com$");
Signature: "RSA-SHA1:213354f9"
```

C. A certificate credential for a specific user whose email address is mab@keynote.research.att.com and whose name, if present, must be "M. Blaze". The credential was issued by the ‘Alice’ authority (whose key is certified in Example B above):

```
KeyNote-Version: 2
Authorizer: "DSA:4401ff92"  # the Alice CA
Licensees: "DSA:12340987"   # mab’s key
Conditions: ((app_domain == "RFC822-EMAIL") &&
             (name == "M. Blaze" || name == "") &&
             (address == "mab@keynote.research.att.com"));
Signature: "DSA-SHA1:ab23487"
```

D. Another certificate credential for a specific user, also issued by the 'Alice' authority. This example allows three different keys to sign as jf@keynote.research.att.com (each for a different cryptographic algorithm). This is, in effect, three credentials in one:

```
KeyNote-Version: "2"
Authorizer: "DSA:4401ff92"   # the Alice CA
Licensees: "DSA:abc991" ||   # jf's DSA key
           "RSA:cde773" ||   # jf's RSA key
           "BFIK:fd091a"     # jf's BFIK key
Conditions: ((app_domain == "RFC822-EMAIL") &&
             (name == "J. Feigenbaum" || name == "") &&
             (address == "jf@keynote.research.att.com"));
Signature: "DSA-SHA1:8912aa"
```

Observe that under policy A and credentials B, C and D, the following action attribute sets are accepted (they return _MAX_TRUST):

```
   _ACTION_AUTHORIZERS = "dsa:12340987"
   app_domain = "RFC822-EMAIL"
   address = "mab@keynote.research.att.com"
and
   _ACTION_AUTHORIZERS = "dsa:12340987"
   app_domain = "RFC822-EMAIL"
   address = "mab@keynote.research.att.com"
   name = "M. Blaze"
```

while the following are not accepted (they return _MIN_TRUST):

```
   _ACTION_AUTHORIZERS = "dsa:12340987"
   app_domain = "RFC822-EMAIL"
   address = "angelos@dsl.cis.upenn.edu"
and
   _ACTION_AUTHORIZERS = "dsa:abc991"
   app_domain = "RFC822-EMAIL"
   address = "mab@keynote.research.att.com"
   name = "M. Blaze"
and
   _ACTION_AUTHORIZERS = "dsa:12340987"
   app_domain = "RFC822-EMAIL"
   address = "mab@keynote.research.att.com"
   name = "J. Feigenbaum"
```

2. WORKFLOW/ELECTRONIC COMMERCE

E. A policy that delegates authority for the "SPEND" application domain to RSA key dab212 when the amount given in the "dollars" attribute is less than 10000.

```
Authorizer: "POLICY"
Licensees: "RSA:dab212"  # the CFO's key
Conditions: (app_domain=="SPEND") && (@dollars < 10000);
```

F. RSA key dab212 delegates authorization to any two signers, from a list, one of which must be DSA key feed1234 in the "SPEND" application when @dollars < 7500. If the amount in @dollars is 2500 or greater, the request is approved but logged.

```
KeyNote-Version: 2
Comment: This credential specifies a spending policy
Authorizer: "RSA:dab212"        # the CFO
Licensees: "DSA:feed1234" &&    # The vice president
               ("RSA:abc123" || # middle manager #1
                "DSA:bcd987" || # middle manager #2
                "DSA:cde333" || # middle manager #3
                "DSA:def975" || # middle manager #4
                "DSA:978add")   # middle manager #5
Conditions: (app_domain=="SPEND")  # note nested clauses
              -> { (@(dollars) < 2500)
                     -> _MAX_TRUST;
                   (@(dollars) < 7500)
                     -> "ApproveAndLog";
                 };
Signature: "RSA-SHA1:9867a1"
```

G. According to this policy, any two signers from the list of managers will do if @(dollars) < 1000:

```
KeyNote-Version: 2
Authorizer: "POLICY"
Licensees: 2-of("DSA:feed1234", # The VP
                "RSA:abc123",   # Middle management clones
                "DSA:bcd987",
                "DSA:cde333",
                "DSA:def975",
                "DSA:978add")
Conditions: (app_domain=="SPEND") &&
            (@(dollars) < 1000);
```

H. A credential from dab212 with a similar policy, but only one signer is required if @(dollars) < 500. A log entry is made if the amount is at least 100.

```
KeyNote-Version: 2
Comment: This one credential is equivalent to six separate
         credentials, one for each VP and middle manager.
         Individually, they can spend up to $500, but if
         it's $100 or more, we log it.
Authorizer: "RSA:dab212"      # From the CFO
Licensees: "DSA:feed1234" ||  # The VP
           "RSA:abc123" ||    # The middle management clones
           "DSA:bcd987" ||
           "DSA:cde333" ||
           "DSA:def975" ||
           "DSA:978add"
Conditions: (app_domain="SPEND")  # nested clauses
              -> { (@(dollars) < 100) -> _MAX_TRUST;
                   (@(dollars) < 500) -> "ApproveAndLog";
                 };
Signature: "RSA-SHA1:186123"
```

Assume a query in which the ordered set of Compliance Values is {"Reject", "ApproveAndLog", "Approve"}. Under policies E and G, and credentials F and H, the Policy Compliance Value is "Approve" (_MAX_TRUST) when:

```
     _ACTION_AUTHORIZERS = "DSA:978add"
     app_domain = "SPEND"
     dollars = "45"
     unmentioned_attribute = "whatever"
 and
     _ACTION_AUTHORIZERS = "RSA:abc123,DSA:cde333"
     app_domain = "SPEND"
     dollars = "550"
```

The following return "ApproveAndLog":

```
     _ACTION_AUTHORIZERS = "DSA:feed1234,DSA:cde333"
     app_domain = "SPEND"
     dollars = "5500"
 and
     _ACTION_AUTHORIZERS = "DSA:cde333"
     app_domain = "SPEND"
     dollars = "150"
```

However, the following return "Reject" (_MIN_TRUST):

```
     _ACTION_AUTHORIZERS = "DSA:def975"
     app_domain = "SPEND"
     dollars = "550"
 and
     _ACTION_AUTHORIZERS = "DSA:cde333,DSA:978add"
     app_domain = "SPEND"
     dollars = "5500"
```

7. Trust-Management Architecture

KeyNote provides a simple mechanism for describing security policy and representing credentials. It differs from traditional certification systems in that the security model is based on binding keys to predicates that describe what the key is authorized by policy to do, rather than on resolving names. The infrastructure and architecture to support a KeyNote system is therefore rather different from that required for a name-based certification scheme. The KeyNote trust-management architecture is based on that of PolicyMaker [BFL96,BFS98].

It is important to understand the separation between the responsibilities of the KeyNote system and those of the application and other support infrastructure. A KeyNote compliance checker will determine, based on policy and credential assertions, whether a proposed action is permitted according to policy. The usefulness of KeyNote output as a policy enforcement mechanism depends on a number of factors:

* The action attributes and the assignment of their values must reflect accurately the security requirements of the application. Identifying the attributes to include in the action attribute set is perhaps the most important task in integrating KeyNote into new applications.

* The policy of the application must be correct and well-formed. In particular, trust must be deferred only to principals that should, in fact, be trusted by the application.

* The application itself must be trustworthy. KeyNote does not directly enforce policy; it only provides advice to the applications that call it. In other words, KeyNote assumes that the application itself is trusted and that the policy assertions it specifies are correct. Nothing prevents an application from submitting misleading or incorrect assertions to KeyNote or from ignoring KeyNote altogether.

It is also up to the application (or some service outside KeyNote) to select the appropriate credentials and policy assertions with which to run a particular query. Note, however, that even if inappropriate credentials are provided to KeyNote, this cannot result in the approval of an illegal action (as long as the policy assertions are correct and the the action attribute set itself is correctly passed to KeyNote).

KeyNote is monotonic; adding an assertion to a query can never result in a query's having a lower compliance value that it would have had without the assertion. Omitting credentials may, of course, result in legal actions being disallowed. Selecting appropriate credentials (e.g., from a distributed database or 'key server') is outside the scope of the KeyNote language and may properly be handled by a remote client making a request, by the local application receiving the request, or by a network-based service, depending on the application.

In addition, KeyNote does not itself provide credential revocation services, although credentials can be written to expire after some date by including a date test in the predicate. Applications that require credential revocation can use KeyNote to help specify and implement revocation policies. A future document will address expiration and revocation services in KeyNote.

Because KeyNote is designed to support a variety of applications, several different application interfaces to a KeyNote implementation are possible. In its simplest form, a KeyNote compliance checker would exist as a stand-alone application, with other applications calling it as needed. KeyNote might also be implemented as a library to which applications are linked. Finally, a KeyNote implementation might run as a local trusted service, with local applications communicating their queries via some interprocess communication mechanism.

8. Security Considerations

Trust management is itself a security service. Bugs in or incorrect use of a KeyNote compliance checker implementation could have security implications for any applications in which it is used.

9. IANA Considerations

This document contains three identifiers to be maintained by the IANA. This section explains the criteria to be used by the IANA to assign additional identifiers in each of these lists.

9.1 app_domain Identifiers

The only thing required of IANA on allocation of these identifiers is that they be unique strings. These strings are case-sensitive for KeyNote purposes, however it is strongly recommended that IANA assign different capitalizations of the same string only to the same organization.

9.2 Public Key Format Identifiers

These strings uniquely identify a public key algorithm as used in the KeyNote system for representing keys. Requests for assignment of new identifiers must be accompanied by an RFC-style document that describes the details of this encoding. Example strings are "rsa-hex:" and "dsa-base64:". These strings are case-insensitive.

9.3 Signature Algorithm Identifiers

These strings uniquely identify a public key algorithm as used in the KeyNote system for representing public key signatures. Requests for assignment of new identifiers must be accompanied by an RFC-style document that describes the details of this encoding. Example strings are "sig-rsa-md5-hex:" and "sig-dsa-sha1-base64:". Note that all such strings must begin with the prefix "sig-". These strings are case-insensitive.

A. Acknowledgments

We thank Lorrie Faith Cranor (AT&T Labs - Research) and Jonathan M. Smith (University of Pennsylvania) for their suggestions and comments on earlier versions of this document.

B. Full BNF (alphabetical order)

```
<ALGORITHM>:: {see section 4.4.2} ;

<Assertion>:: <VersionField>? <AuthField> <LicenseesField>?
              <LocalConstantsField>? <ConditionsField>?
              <CommentField>? <SignatureField>? ;

<Assignments>:: "" | <AttributeID> "=" <StringLiteral> <Assignments>
;

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

<AuthField>:: "Authorizer:" <AuthID> ;

<AuthID>:: <PrincipalIdentifier> | <DerefAttribute> ;

<Clause>:: <Test> "->" "{" <ConditionsProgram> "}"
         | <Test> "->" <Value> | <Test> ;

<Comment>:: "#" {ASCII characters} ;

<CommentField>:: "Comment:" {Free-form text} ;

<ConditionsField>:: "Conditions:" <ConditionsProgram> ;

<ConditionsProgram>:: "" | <Clause> ";" <ConditionsProgram> ;

<DerefAttribute>:: <AttributeID> ;

<ENCODEDBITS>:: {see section 4.4.2} ;

<FloatEx>:: <FloatEx> "+" <FloatEx> | <FloatEx> "-" <FloatEx>
          | <FloatEx> "*" <FloatEx> | <FloatEx> "/" <FloatEx>
          | <FloatEx> "^" <FloatEx> | "-" <FloatEx>
          | "(" <FloatEx> ")" | <FloatLiteral> | "&" <StrEx> ;

<FloatRelExpr>:: <FloatEx> "<" <FloatEx> | <FloatEx> ">" <FloatEx>
               | <FloatEx> "<=" <FloatEx>
               | <FloatEx> ">=" <FloatEx> ;
```

```
   <FloatLiteral>:: <IntegerLiteral>"."<IntegerLiteral> ;

   <IDString>:: <ALGORITHM>":"<ENCODEDBITS> ;

   <IntegerLiteral>:: {Decimal number of at least one digit} ;

   <IntEx>:: <IntEx> "+" <IntEx> | <IntEx> "-" <IntEx>
           | <IntEx> "*" <IntEx> | <IntEx> "/" <IntEx>
           | <IntEx> "%" <IntEx> | <IntEx> "^" <IntEx>
           | "-" <IntEx> | "(" <IntEx> ")" | <IntegerLiteral>
           | "@" <StrEx> ;

   <IntRelExpr>:: <IntEx> "==" <IntEx> | <IntEx> "!=" <IntEx>
                | <IntEx> "<" <IntEx>  | <IntEx> ">" <IntEx>
                | <IntEx> "<=" <IntEx> | <IntEx> ">=" <IntEx> ;

   <K>:: {Decimal number starting with a digit from 1 to 9} ;

   <KeyID>:: <StrEx> ;

   <LicenseesExpr>:: "" | <PrincExpr> ;

   <LicenseesField>:: "Licensees:" <LicenseesExpr> ;

   <LocalConstantsField>:: "Local-Constants:" <Assignments> ;

   <OpaqueID>:: <StrEx> ;

   <PrincExpr>:: "(" <PrincExpr> ")" | <PrincExpr> "&&" <PrincExpr>
               | <PrincExpr> "||" <PrincExpr>
               | <K>"-of(" <PrincList> ")" | <PrincipalIdentifier>
               | <DerefAttribute> ;

   <PrincipalIdentifier>:: <OpaqueID> | <KeyID> ;

   <PrincList>:: <PrincipalIdentifier> | <DerefAttribute>
               | <PrincList> "," <PrincList> ;

   <RegExpr>:: {POSIX 1003.2 Regular Expression}

   <RelExpr>:: "(" <RelExpr> ")" | <RelExpr> "&&" <RelExpr>
             | <RelExpr> "||" <RelExpr> | "!" <RelExpr>
             | <IntRelExpr> | <FloatRelExpr> | <StringRelExpr>
             | "true" | "false" ;

   <Signature>:: <StrEx> ;

   <SignatureField>:: "Signature:" <Signature> ;
```

```
   <StrEx>:: <StrEx> "." <StrEx> | <StringLiteral> | "(" <StrEx> ")"
           | <DerefAttribute> | "$" <StrEx> ;

   <StringLiteral>:: {see section 4.3.1} ;

   <StringRelExpr>:: <StrEx> "==" <StrEx> | <StrEx> "!=" <StrEx>
                   | <StrEx> "<" <StrEx> | <StrEx> ">" <StrEx>
                   | <StrEx> "<=" <StrEx> | <StrEx> ">=" <StrEx>
                   | <StrEx> "~=" <RegExpr> ;

   <Test>:: <RelExpr> ;

   <Value>:: <StrEx> ;

   <VersionField>:: "KeyNote-Version:" <VersionString> ;

   <VersionString>:: <StringLiteral> | <IntegerLiteral> ;
```

References

[BFL96] M. Blaze, J. Feigenbaum, J. Lacy. Decentralized Trust Management. Proceedings of the 17th IEEE Symp. on Security and Privacy. pp 164-173. IEEE Computer Society, 1996. Available at <ftp://ftp.research.att.com/dist/mab/policymaker.ps>

[BFS98] M. Blaze, J. Feigenbaum, M. Strauss. Compliance-Checking in the PolicyMaker Trust-Management System. Proc. 2nd Financial Crypto Conference. Anguilla 1998. LNCS #1465, pp 251-265, Springer-Verlag, 1998. Available at <ftp://ftp.research.att.com/dist/mab/pmcomply.ps>

[Bla99] M. Blaze, J. Feigenbaum, J. Ioannidis, A. Keromytis. The Role of Trust Management in Distributed System Security. Chapter in Secure Internet Programming: Security Issues for Mobile and Distributed Objects (Vitek and Jensen, eds.). Springer-Verlag, 1999. Available at <ftp://ftp.research.att.com/dist/mab/trustmgt.ps>.

[Cro82] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.

[DSA94] Digital Signature Standard. FIPS-186. National Institute of Standards, U.S. Department of Commerce. May 1994.

[PKCS1] PKCS #1: RSA Encryption Standard, Version 1.5. RSA Laboratories. November 1993.

[RSA78] R. L. Rivest, A. Shamir, L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, v21n2. pp 120-126. February 1978.

Authors' Addresses

Comments about this document should be discussed on the keynote-users mailing list hosted at nsa.research.att.com. To subscribe, send an email message containing the single line

subscribe keynote-users

in the message body to <majordomo@nsa.research.att.com>.

Questions about this document can also be directed to the authors as a group at the keynote@research.att.com alias, or to the individual authors at:

Matt Blaze
AT&T Labs - Research
180 Park Avenue
Florham Park, New Jersey 07932-0971

EMail: mab@research.att.com

Joan Feigenbaum
AT&T Labs - Research
180 Park Avenue
Florham Park, New Jersey 07932-0971

EMail: jf@research.att.com

John Ioannidis
AT&T Labs - Research
180 Park Avenue
Florham Park, New Jersey 07932-0971

EMail: ji@research.att.com

Angelos D. Keromytis
Distributed Systems Lab
CIS Department, University of Pennsylvania
200 S. 33rd Street
Philadelphia, Pennsylvania 19104-6389

EMail: angelos@dsl.cis.upenn.edu

Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

tmipsec.pdf is appearing in NDSS 2001.