SPKI Working Group Matt Blaze
Internet Draft Joan Feigenbaum
expires in six months Angelos D. Keromytis
April 1998

The KeyNote Trust-Management System
draft-angelos-spki-keynote-01.txt (A)

Status of this Memo

Abstract

This memo describes KeyNote, a simple trust-management system to support public-key infrastructure. It outlines the syntax and semantics of keynote credentials, describes action environment processing, and describes the application architecture into which a KeyNote implementation would fit.

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [Bra97].

1. Introduction

This memo describes KeyNote, a simple trust-management system for public-key infrastructures. Trust management, introduced in the PolicyMaker system [BFL96], is a unified approach to specifying and interpreting security policies, credentials, and relationships that allows direct authorization of security-critical actions. In particular, a trust-management system combines the notion of specifying security policy with the mechanism for specifying security credentials (subsuming the role of "certificates"). Credentials describe a specific delegation of trust among public keys; unlike traditional certificates, which bind keys to names, trust-management credentials bind keys to the authorization to perform specific tasks.

KeyNote provides a simple notation for specifying both local security policies and security credentials that can be sent over an untrusted network. Policies and credentials, called "assertions" as in PolicyMaker, contain predicates that describe the trusted actions permitted by the holders of specific public keys. A signed assertion that can be sent over an untrusted network is called a Credential Assertion. Credential assertions, which serve the role of "certificates," have the same syntax as policy assertions with the additional feature that they are signed by the entity delegating the trust. A KeyNote evaluator accepts as input a set of local policy assertions, a collection of credential assertions, and a collection of attributes, called an "action environment," that describes a proposed trusted action associated with a set of public keys. KeyNote determines whether proposed actions are consistent with local policy by applying the assertion predicates to the action environment.

Although the basic design of KeyNote is similar in spirit to that of PolicyMaker, KeyNote's features have been simplified to more directly support public-key infrastructure-like applications. The central differences between PolicyMaker and KeyNote are:

- KeyNote predicates are written in a simple notation based on C-like expressions and regular expressions.
- The KeyNote system always returns a boolean (trusted or not) answer.
- Credential signature verification is built in to the KeyNote system.
- Assertion syntax is based on a human-readable "RFC-822"-style syntax.
- Trusted actions are described by simple attribute/value pairs.

2. KeyNote Assertion Format

All KeyNote assertions are encoded in ASCII strings. Four mandatory fields MUST appear in all assertions (KEYNOTE-VERSION, SIGNER, KEY-PREDICATE, and ACTION-PREDICATE); three optional fields MAY appear (COMMENT, SIGNATURE, and LOCAL-INIT).

The version of KeyNote assertions described in this document is 1. All fields MUST start at the beginning of a line; fields may be continued by indenting with at least one SPACE or TAB character at the beginning of the line. Whitespace separates tokens but is otherwise ignored outside of quoted strings (our grammars below omit whitespace processing in the interest of readability). All name tokens are case-insensitive. It is encouraged that keys be encoded in lower-case hex digits. String comparison of keys for internal purposes (e.g., matching a key in the KEY-PREDICATE of one assertion with a key in the SIGNER field of another assertion) MUST be case insensitive. In the following sections, the notation [X]* means zero or more repetitions of the string X. The notation [X]+ means one or more repetitions of the string X.

2.1 Key Encoding

Keys are encoded as ALG[:LEN:HEXENC]+, where LEN is an ASCII-encoded decimal number, indicating the number of characters in the HEXENC field. HEXENC is key encoded in hexadecimal digits. If more than one component is needed by the signature algorithm, the extra components are appended. The required number of components is determined by the key type. ALG is an ASCII string that describes the key type (such as RSA or DSA).

2.2 The KEYNOTE-VERSION field

The KEYNOTE-VERSION field is of the form KEYNOTE-VERSION:VerNumber, where VerNumber is an ASCII-encoded decimal number. The current version is 1. This field MUST be the first field appearing in a KeyNote assertion.

2.3 The LOCAL-INIT field

The initialization field is of the form:
Local-Init: Name = ConstantString [, Name = ConstantString]*

Name is an identifier that can be used in the ACTION-PREDICATE and KEY-PREDICATE fields instead of the string ConstantString. If the LOCAL-INIT field defines more than one identifier, it can occupy more than one line and be indented. ConstantString can be composed by concatenating smaller strings using the "+" operator (see the examples). If an initialization identifier is accessed but has not been defined, it should evaluate to the empty string. The initialized identifier ANGELOS_DSA_KEY could, for example, be used in the KEY-PREDICATE field as KEY-PREDICATE: $ANGELOS_DSA_KEY

When an initialization identifier is accessed in an ACTION-PREDICATE expression, it shadows the value of any action environment identifier, for this assertion only. If an identifier is declared more than once in the LOCAL-INIT field, the assertion MUST be considered invalid.

2.4 The SIGNER field

The SIGNER field is of the form SIGNER:KEY, where KEY is a key encoded as described in 2.1 or an initialization constant as described in 2.3. The SIGNER field may instead be of the form SIGNER:Policy. A Policy assertion is one that is trusted directly by the local environment. It can serve as the "root" of a trust structure that authorizes a requested action. A valid input to the KeyNote evaluator must contain at least one Policy assertion. Because they are trusted locally, Policy assertions do not require cryptographic signature verification.

2.5 The KEY-PREDICATE field

The KEY-PREDICATE field is of the form Key-Predicate:KEY-EXPR. KEY-EXPR is given by the following grammar:

```
KEY-EXPR: (KEY-EXPR) |
          KEY-EXPR "&&" KEY-EXPR |
          KEY-EXPR "||" KEY-EXPR |
          K-of(KEYLIST) |
          KEY |
          "$"STRING

KEYLIST:  KEY |
          "$"STRING |
          KEY, KEYLIST |
          "$"STRING, KEYLIST

STRING: [a-zA-Z0-9][a-zA-Z0-9_\.]*
```

The "&&" operator has higher precedence than the "||" operator.

"K" is an ASCII-encoded decimal number. A KEYLIST SHOULD contain at least K keys. Whitespace characters (space, tab, newline) in KEY-EXPR MUST be ignored. The semantics of k-OF are that at least K distinct keys from the KEYLIST must authorize a request. If the KEY-EXPR field is empty, it always evaluates to TRUE and is used for direct authorization of a ACTION-PREDICATE by a policy or a credential.

2.6 The SIGNATURE field

The SIGNATURE field is of the form Signature:NAME[:LEN:SIG]+, where NAME is an ASCII string that indicates the signature type (e.g., RSA-MD5-PKCS1). LEN is an ASCII encoded decimal number that indicates the length of the SIG field. SIG is the signature encoded in hexadecimal digits. If more than one component is needed by the signature algorithm, the extra components are appended. The required number of components is determined by the NAME field value. The signature is computed over the KEYNOTE-VERSION, SIGNER, LOCAL-INIT, KEY-PREDICATE, COMMENT, and ACTION-PREDICATE fields, concatenated with the NAME field, as they appear in the credential. This field MUST be last in a KeyNote assertion.

2.7 The COMMENT field

The COMMENT field is of the form Comment: .*
The interpretation of this field is application-dependent.

2.8 The ACTION-PREDICATE field

The ACTION-PREDICATE field is of the form ACTION-PREDICATE: EXPR, where EXPR is described by the following grammar:

```
EXPR: "(" EXPR ")" |
      EXPR "&&" EXPR |       /* Logical AND */
      EXPR "||" EXPR |       /* Logical OR */
      "!"EXPR |              /* Logical NOT */
      NUMEXPR |
      FLOATEXPR |
      STRINGEXPR |
      true | false

NUMEXPR: NUMEX < NUMEX |     /* Integer expression comparisons */
         NUMEX > NUMEX |
         NUMEX <= NUMEX |
         NUMEX >= NUMEX |
         NUMEX == NUMEX |
         NUMEX != NUMEX

FLOATEXPR: FLOATEX < FLOATEX | /* Floating point */
           FLOATEX > FLOATEX |
           FLOATEX <= FLOATEX |
           FLOATEX >= FLOATEX

STRINGEXPR: STR == STR |      /* String comparisons */
            STR != STR |
            STR < STR |      /* Alphanumeric comparison */
            STR > STR |
            STR <= STR |
            STR >= STR |
            STR ~= REGEXP    /* Regular expression matching */

STR: STR + STR |             /* String concatenation */
     STR . STR |             /* Also string concatenation */
     LITERALSTRING |
     VARIABLE
     "$(" STR ")"

NUMEX: NUMEX + NUMEX |       /* Arithmetic operations */
```

```
          NUMEX - NUMEX |
          NUMEX * NUMEX |
          NUMEX / NUMEX |
```

```
        NUMEX % NUMEX |
        NUMEX ^ NUMEX |       /* Exponentiation */
        -NUMEX
        (NUMEX) |
        NUMBER |
        NUMVARIABLE

FLOATEX: FLOATEX + FLOATEX |       /* Floating point operations */
         FLOATEX - FLOATEX |
         FLOATEX * FLOATEX |
         FLOATEX / FLOATEX |
         FLOATEX ^ FLOATEX |       /* Exponentiation */
         -FLOATEX
         (FLOATEX) |
         FLOAT |
         FLOATVARIABLE

NUMBER: [0-9]+
FLOAT: {NUMBER} "." {NUMBER}
STRING: [a-zA-Z][a-zA-Z0-9_]*
VSTRING: [a-zA-Z0-9][a-zA-Z0-9_]*
DQUOTE: \"
LITERALSTRING: DQUOTE (([^\"\n])|(\\[.\n]))* DQUOTE
VARIABLE: "$"VSTRING
NUMVARIABLE: "@"VSTRING
FLOATVARIABLE: "&"VSTRING
```

The numeric operation precedence is (higher to lower) ^, (*, /, %), (+, -). Operators of equal precedence are evaluated left-to-right. REGEXP refers to the POSIX 1003.2 regular expression syntax and semantics. Single-backslash ("\") elimination must be performed on LITERALSTRINGs (e.g., "\ac" becomes "ac"). If an ACTION-PREDICATE field is empty, it always evaluates to TRUE. A division (or modulo) by zero cause the enclosing boolean expression to evaluate to false. String operations (including regexps) can be case-sensitive or case-insensitive, specified as a run-time option.

3. ACTION ENVIRONMENTS

Trusted actions to be evaluated by KeyNote are described by a collection of attribute/value pairs called the Action Environment. An action environment is passed to the KeyNote system as part of each query and provides the values of the variables used by assertion predicates.

The action environment specifies a collection of values of named attributes. The attributes may be examined as variables by KeyNote trust predicates.

The exact format for specifying an action environment is determined by the particular KeyNote implementation. In general, an environment may be thought of as a list of assignments to variables:

```
ATTRIBUTE1=VALUE1
ATTRIBUTE2=VALUE2
...
```

If an action environment attribute is not defined, it MUST evaluate to the empty string (if accessed as a string) or the value zero (if accessed as an integer or a float).

An attribute that is accessed as an integer (by prepending the "@" sign) MUST contain only digits. Similarly, an attribute that is accessed as a float (through the "&" sign) MUST consist entirely of digits and at most one period. In both cases if the attribute contains any illegal character, the returned value MUST be zero.

Only one attribute, called "$ACTION_SIGNERS", is mandatory. This attribute lists the key or keys associated with the action environment (e.g., signer of the message whose trust is being evaluated by KeyNote). The $ACTION_SIGNERS attribute is used to provide the initial keys to match against the KEY-PREDICATEs.

In most cases, the $ACTION_SIGNERS attribute will consist of a single public key (of the ALG:LEN:VAL form specified in the previous section):

```
$ACTION_SIGNERS="dsa:12:abcdef123456"
```

If an action is associated with more than one key, the ACTION_SIGNERS attribute may contain a comma-separated list of keys.

```
$ACTION_SIGNERS="dsa:20:123400000123abd12212,
                 rsa:28:ab34781acc01923cdeff232ff232"
```

The names of all other attributes in the action environment are not specified by KeyNote but must be agreed upon by the writers of any policies and credentials that are to cooperate in a KeyNote query evaluation. By convention, the name of the application domain in which environment attributes should be interpreted is specified in the attribute APP_DOMAIN. The IANA will provide a registry of reserved APP_DOMAIN names with the names and meanings of the attributes they use. Note that an attribute with a particular name may have different meanings in different application domains. Note that the use of the registry is optional; a policy or credential may depend on any attribute names used by the credentials to which trust is deferred.

For example, an email application might reserve the APP_DOMAIN

"RFC822-EMAIL" and might use the following attributes:

$ADDRESS (the email address of a message's sender)

$NAME (the human name of the message sender)
$ORGANIZATION (the organization name).

The values of these attributes may be derived in the obvious way from the email message headers.

Note that RFC822-EMAIL is simply 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 suggests.)

4. KeyNote Action Evaluation

This section describes the semantics of KeyNote action evaluation. An implementation is free to use any algorithm that provides equivalent semantics.

Initialization:

The variable $App_Domain is assigned the name of the application (e.g., "RFC822-EMAIL").

The keys that authenticate the request for a trusted action are assigned to the variable $Action_Signers.

The rest of the action environment attributes are placed in their respective variables.

The time of day MAY be placed in the variables $GMTTimeOfDay and $LocalTimeOfDay, using the format YYYYMMDDHHMMSS (e.g., 19980307191512).

Any other implementation-dependent variables and their bindings are also created at this step.

For each KeyNote assertion passed to the evaluation engine, the following steps are taken:

The ACTION-PREDICATE expression is evaluated. If the result is boolean TRUE, and the key expression in the KEY-PREDICATE field is also true, the request is approved. Otherwise, it is rejected.

The KEY-PREDICATE field public-key expression is evaluated as follows:

Let the key expression contain public key PK_i. A boolean variable `PK_i' corresponds to this key.

If there is no assertion in which PK_i is the SIGNER, then the boolean variable `PK_i' is false.

If there is at least one assertion in which PK_i is the source, then the boolean variable `PK_i' is true if and only if at least one of those assertions is true.

For a request to be approved, there must exist a directed graph of KeyNote assertions rooted at a POLICY assertion of the evaluator that evaluates to TRUE. If such a directed graph cannot be constructed, the request is denied for lack of authorization. Delegation of some authorization from key A to a set of keys B is expressed as an assertion with key A in the SIGNER field, key set B in the KEY-PREDICATE field, and the authorization delegated encoded in the ACTION-PREDICATE field. How the expression digraph is constructed is implementation-dependent, in particular because different implementations may use different algorithms for optimizing the graph construction.

5. Examples

In the interest of readability, these examples use much shorter keys than would ordinarily be used. 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 delegating trust to the holder of RSA key abc123:

```
KEYNOTE-VERSION: 1
SIGNER: policy
KEY-PREDICATE: rsa-sha1-pkcsX:6:abc123
ACTION-PREDICATE: true
```

   B. A credential assertion in which RSA Key abc123 trusts either RSA key 4401ff92 or DSA key d1234f 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 4401ff92 and d1234f as certification authorities for the keynote.research.att.com domain.

```
KEYNOTE-VERSION: 1
LOCAL-INIT: TRUSTED_PARTY1="dsa-sha1-pkcsX:8:4401" + "ff92",
            TRUSTED_PARTY2="rsa-sha1-pkcsX:6:d1234f"
SIGNER: rsa-sha1-pkcsX:6:abc123
KEY-PREDICATE: $TRUSTED_PARTY1 || $TRUSTED_PARTY2
ACTION-PREDICATE: ($app_domain == "RFC822-EMAIL") &&
```

```
                        ($address ~=
                            "^.*@keynote\\.research\\.att\\.com$")
           SIGNATURE: rsa-md5-pkcsX:8:213354f9
```

C. A certificate credential for a specific user, issued by one of the certification authorities above:

```
KEYNOTE-VERSION: 1
SIGNER: dsa-sha1-pkcsX:8:4401ff92
KEY-PREDICATE: dsa-sha1-pkcsX:8:12340987
ACTION-PREDICATE: (($app_domain == "RFC822-EMAIL") &&
                   ($name == "M. Blaze" || $name == "") &&
                   ($address ==
                            "mab@keynote.research.att.com"))
SIGNATURE: dsa-sha1-pkcsX:8:ab23487
```

D. Another certificate credential for a specific user, issued by one of the certification authorities above. This one allows three different keys to sign as jf@keynote.research.att.com (each with a different crypto algorithm). Three credentials in one:

```
KEYNOTE-VERSION: 1
SIGNER: dsa-sha1-pkcsX:8:4401ff92=
KEY-PREDICATE: dsa-sha1-pkcsX:6:abc991 ||
               rsa-sha1-pkcsX:6:cde773 ||
               rsa-md5-pkcsX:6:fd091a
ACTION-PREDICATE: (($app_domain == "RFC822-EMAIL") &&
               ($name == "J. Feigenbaum" || $name == "") &&
               ($address == "jf@keynote.research.att.com"))
SIGNATURE: dsa-sha1-pkcsX:8:8912aa
```

Observe that under policy A and credentials B, C and D, the following action environments are accepted:

```
$action_signer = "dsa-sha1-pkcsX:8:12340987"
$app_domain = "RFC822-EMAIL"
$address = "mab@keynote.research.att.com"
```

and

```
$action_signer = "dsa-sha1-pkcsX:8:12340987"
$app_domain = "RFC822-EMAIL"
$address = "mab@keynote.research.att.com"
$name = "M. Blaze"
```

while the following are not accepted:

```
$action_signer = "dsa-sha1-pkcsX:8:12340987"
$app_domain = "RFC822-EMAIL"
$address = "angelos@dsl.cis.upenn.edu"
```

and

```
        $action_signer = "dsa-sha1-pkcsX:6:abc991"
        $app_domain = "RFC822-EMAIL"
```

```
    $address = "mab@keynote.research.att.com"
    $name = "M. Blaze"
```

and

```
    $action_signer = "dsa-sha1-pkcsX:8:12340987"
    $app_domain = "RFC822-EMAIL"
    $address = "mab@keynote.research.att.com"
    $name = "J. Feigenbaum"
```

E. Here's a credential that does not allow delegation to another key:

```
    KEYNOTE-VERSION: 1
    LOCAL-INIT: KEY1="dsa-sha1-pkcsX:6:fde995"
    SIGNER: dsa-sha1-pkcsX:8:4401ff92
    KEY-PREDICATE: $KEY1
    ACTION-PREDICATE:($app_domain="RFC822-EMAIL") &&
                     ($action_signers=$KEY1) &&
                     ($name == "A. Keromytis" ||
                      $name == "") &&
                     ($address ==
                           "angelos@keynote.research.att.com")
    SIGNATURE: dsa-sha1-pkcsX:8:fed121ab
```

Now, even if we add a credential:

```
    KEYNOTE-VERSION: 1
    SIGNER: dsa-sha1-pkcsX:6:fde995
    KEY-PREDICATE: dsa-sha1-pkcsX:8:bad22bad
    ACTION-PREDICATE: true
    SIGNATURE: dsa-sha1-pkcsX:6:973cc1
```

we still won't accept this action environment:

```
    $action_signer = "dsa-sha1-pkcsX:8:bad22bad"
    $app_domain = "RFC822-EMAIL"
    $address = "angelos@keynote.research.att.com"
    $name = "A. Keromytis"
```

Although, of course, we would accept:

```
    $action_signer = "dsa-sha1-pkcsX:6:fde995"
    $app_domain = "RFC822-EMAIL"
    $address = "angelos@keynote.research.att.com"
    $name = "A. Keromytis"
```

but not:

```
$action_signer = "dsa-sha1-pkcsX:6:fde995"
$app_domain = "RFC822-EMAIL"
$address = "angelos@keynote.research.at.com"
$name = "Matt Blaze"
```

2. WORKFLOW/ELECTRONIC COMMERCE

F. A policy that delegates authority for the "SPEND" application domain to RSA key babe12 when @dollars is less than 10000.

```
KEYNOTE-VERSION: 1
SIGNER: policy
KEY-PREDICATE: rsa-sha1-pkcsX:6:babe12
ACTION-PREDICATE: ($app_domain="SPEND") && (@dollars < 10000)
```

G. RSA key babe12 requires the signature of at least 2 signers, one of which must be DSA key feed1234 in the "SPEND" application when @dollars is less than 5000

```
KEYNOTE-VERSION: 1
SIGNER: rsa-sha1-pkcsX:6:babe12
KEY-PREDICATE: dsa-sha1-pkcsX:8:feed1234 &&
               (rsa-sha1-pkcsX:6:abc113 ||
                dsa-sha1-pkcsX:6:bcd987 ||
                dsa-sha1-pkcsX:6:cde333 ||
                dsa-sha1-pkcsX:6:def975 ||
                dsa-sha1-pkcsX:6:978add)
ACTION-PREDICATE: ($app_domain="SPEND") &&
                  (@dollars < 5000)
SIGNATURE: rsa-sha1-pkcsX:6:9867a1
```

H. Any two signers if @dollars < 1000:

```
KEYNOTE-VERSION: 1
SIGNER: policy
KEY-PREDICATE: 2-of(dsa-sha1-pkcsX:8:feed1234:,
                    rsa-sha1-pkcsX:6:abc113,
                    dsa-sha1-pkcsX:6:bcd987,
                    dsa-sha1-pkcsX:6:cde333,
                    dsa-sha1-pkcsX:6:def975,
                    dsa-sha1-pkcsX:6:978add)
ACTION-PREDICATE: ($app_domain="SPEND") &&
                  (@dollars < 1000)
```

I. As above, but only one signer required if @dollars < 100.

```
KEYNOTE-VERSION: 1
SIGNER: rsa-sha1-pkcsX:6:babe12
KEY-PREDICATE: (dsa-sha1-pkcsX:8:feed1234 ||
                rsa-sha1-pkcsX:6:abc123 ||
                dsa-sha1-pkcsX:6:bcd987 ||
                dsa-sha1-pkcsX:6:cde333 ||
                dsa-sha1-pkcsX:6:def975 ||
```

```
                    dsa-sha1-pkcsX:6:978add)
        ACTION-PREDICATE: ($app_domain="SPEND") &&
                          (@dollars < 100)
```

```
SIGNATURE: rsa-sha1-pkcsX:6:786123
```

Under policies F and H, and credentials G and I, we accept:

```
$action_signer = "dsa-sha1-pkcsX:6:978add"
$app_domain = "SPEND"
@dollars = 45
$unmentioned_variable="whatever"
```

and

```
$action_signer = "rsa-sha1-pkcsX:6:abc123,
                  dsa-sha1-pkcsX:8:cde333"
$app_domain = "SPEND"
@dollars = 550
```

and

```
$action_signer = "dsa-sha1-pkcsX:8:feed1234,
                  dsa-sha1-pkcsX:6:cde333"
$app_domain = "SPEND"
@dollars = 2500
```

and

```
$action_signer = "rsa-sha1-pkcsX:8:babe12"
$app_domain = "SPEND"
@dollars = 2000
```

However, the following are not accepted:

```
$action_signer = "dsa-sha1-pkcsX:6:def975"
$app_domain = "SPEND"
@dollars = 550
```

and

```
$action_signer = "dsa-sha1-pkcsX:8:cde333,
                  dsa-sha1-pkcsX:8:978add"
$app_domain = "SPEND"
@dollars = 2500
```

3. COMMAND AND CONTROL AUTHORIZATION

J. A policy that at least two signers are required to authorize the launch of missiles against London or Moscow.

```
KEYNOTE-VERSION: 1
```

```
        SIGNER: policy
        KEY-PREDICATE: 2-of(dsa-sha1-pkcsX:8:badfeed1,
                            rsa-sha1-pkcsX:8:ff123ad3,
```

```
                              dsa-sha1-pkcsX:8:198714fd,
                              dsa-sha1-pkcsX:8:a1984cde,
                              dsa-sha1-pkcsX:6:975135)
           ACTION-PREDICATE: ($app_domain="NUKE") &&
                            ($action="launch") &&
                            ($delivery_system="missile") &&
                            (($target="moscow") || ($target="london"))
```

6. 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 than that for a name-based certification scheme. The KeyNote trust-management architecture is based on that of PolicyMaker [BFL96].

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 evaluator will determine, based on policy and credential assertions, whether a proposed action is permitted according to policy. The usefulness of this determination depends on a number of factors. First, the action environment attributes and the assignment of their values must reflect accurately the security requirements of the application. Identifying the attributes to include in the action environment is perhaps the most important task in integrating KeyNote into new applications. Second, the policy of the application must be correct and well-formed. In particular, trust must be deferred only to keys and for predicates that should, in fact, be trusted by the application. Finally, KeyNote does not directly enforce policy; it only provides advice to the applications that call it. KeyNote assumes that the application itself is trusted and that the policy assertions are correct. Nothing prevents an application from submitting misleading assertions to KeyNote, or from ignoring KeyNote altogether.

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 that even if inappropriate credentials are provided to KeyNote, this cannot result in the approval of an illegal action environment (as long as the policy assertions are correct and the the action environment itself is correctly passed to KeyNote). KeyNote is monotonic; adding an assertion to a query can never result in a query being rejected if it would have been accepted without the assertion. Omitting credentials may, of course, result in legal action environments being disallowed. Selecting appropriate credentials (e.g., from a distributed database or "key server") is outside the scope of KeyNote itself, and may properly be handled by the remote client making a request, by the local machine verifying 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 draft will address expiration and revocation services in KeyNote.

Observe that KeyNote adopts an almost opposite approach from that of a general-purpose name-based certification scheme. In name-based schemes (such as X.509), the infrastructure aims to provide a common application-independent certificate, with each application left to develop its own mechanism to interpret the security semantics of the name. KeyNote, on the other hand, aims to provide a common, application-independent mechanism for use with application-specific credentials and policies. Each application (or class of applications) will develop its own set of attributes, with application-specific credentials and policies created to operate on them.

Nevertheless, it is possible to take advantage of an existing general-purpose name-based infrastructure but still use KeyNote to specify policy and trust in some applications. If an X.509 [X509], PGP [Zim95], or SDSI [LR97] -based certificate distribution infrastructure provides reliable bindings between names and keys, these certificates can be converted to KeyNote assertions that verify that an appropriate action environment attribute has the correct name. Policy assertions can be used to specify the X.509, PGP, or SDSI certification authorities that are trusted for various kinds of names, etc.

Because KeyNote is designed to support a variety of applications, several different application interfaces to a KeyNote implementation are possible. In the simplest, a KeyNote evaluator 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.

7. Security Considerations

This draft discusses a trust-management system for public-key infrastructures. The draft is itself concerned with a security mechanism.

References

[BFL96] M. Blaze, J. Feigenbaum, J. Lacy, Decentralized Trust Management, 1996 IEEE Conference on Privacy and Security, Oakland, 1996.

[LR97] B. Lampson, R. Rivest, SDSI - Simple Distributed System Infrastructure, Draft, 1997.

[Zim95] P. Zimmermann, PGP User's Manual, 1995.

[Bra97] S. Bradner, Key words for use in RFCs to Indicate Requirement Level, RFC 2119, March 1997.

[X509] CCITT, Recommendation X.509: The Directory Authentication Framework, December 1988.

[SPKI] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, SPKI Certificate Theory, Work in Progress, http://www.clark.net/pub/cme/theory.txt

Contacts

Comments about this document should be discussed on the spki@c2.net mailing list.

Questions about this document can also be directed to:

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

mab@research.att.com

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

jf@research.att.com

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

angelos@dsl.cis.upenn.edu