KeyNote: Trust Management for Public-Key Infrastructures

Matt Blaze[1] Joan Feigenbaum[1] Angelos D. Keromytis[2]

[1] AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932 USA
{mab,jf}@research.att.com

[2] Distributed Systems Lab
CIS Department, University of Pennsylvania
200 S. 33rd Str., Philadelphia, PA 19104 USA
angelos@dsl.cis.upenn.edu

Abstract. This paper discusses the rationale for designing a simple trust-management system for public-key infrastructures, called *KeyNote*. The motivating principles are expressiveness, simplicity, and extensibility. We believe that none of the existing public-key infrastructure proposals provide as good a combination of these three factors.

1 Introduction

Trust management, introduced in the PolicyMaker system [2], is a unified approach to specifying and interpreting security policies, credentials, and relationships that allows direct authorization of security-critical actions. Security credentials (which subsume the role of "certificates") 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," 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 subsume 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 [1] 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.
- KeyNote assertions always return a boolean (authorized 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.

SPKI/SDSI [4] also essentially attempts to do trust-management for public-key infrastructures. We believe that KeyNote provides a better solution, because it was designed to be simpler, more extensible, and more expressive than SPKI (and certainly than X.509 [7]). In the following sections, we expand on these three design principles.

2 Simplicity

Simplicity in the design of KeyNote manifests itself in various aspects of the system:

- Narrow focus.
  KeyNote 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. Other approaches include name-based schemes (such as X.509 [7]), in which 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, and SPKI/SDSI [4], which attempts to provide both an authorization and a naming mechanism. Both systems are unnecessarily large and complex (because secure naming and authorization are orthogonal).
- Easily describable system.
  The basic KeyNote document [1] is 15 pages long, including an extensive examples section. We believe that such a compact document makes the system easy to understand. Constrast this size with other systems' documentation (especially those produced by committees!)
- Easy to understand and create assertions.
  The KeyNote assertion format is human-readabled, ASCII-based. Furthermore, the notation used to describe the trusted actions is based on C-like expressions (arithmetic, string, and boolean operations), which is widely known and used. Although we intend to provide an easy-to-use graphical user interface, it is possible to write a KeyNote assertion using just a text editor (except for the signature of course).

– Easy to implement.

  Our reference implementation is less than 1500 lines of *C* and lex/yacc specification and was written with readalibity in mind. The small code size argues for robust and relatively bug-free implementations. The same cannot be said of all other schemes.

3 Expressiveness

The KeyNote language is designed to make it easy to express and evaluate the kinds of policies and trust delegations that occur in "public-key infrastructure" applications.

KeyNote syntax is minimal and reflects the system's focus on delegation of authority. The basic element of KeyNote programming is the *assertion*. Assertions are the mechanism by which a key (or collection of keys) is authorized to perform various trusted actions. Assertions are used to specify local policy. The same assertion syntax is also used to provide signed credentials in which one party defers authorization to another. Thus security policies and credentials share a common syntax, and policy can be used as a credential simply by signing it. Assertions are written independently from one another and are essentially autonomous programs that do not communicate directly with one another or depend on other assertions or externally-defined data structures.

Assertions are designed to be easy for humans and computers to write and understand. In particular, simple authorizations have simple syntax, and it is usually easy to determine what an assertion does simply by reading it. The function of an assertion is to allow an entity to authorize another entity to perform specific trusted actions. An assertion is divided into sections that reflect the logical components of such an authorization. One section identifies the authorizer (either local policy or, in the case of credentials, the key that signed it). Another section, the "key predicate," describes the key or collection of keys being authorized. Finally, the "action predicate" describes the action being authorized.

Assertions are designed to require only minimal computation to evaluate. Key predicates and authorization predicates are based on simple C-like expressions that match against action environment attributes. In particular, there are no loops or function calls, and an assertion can be parsed in a single pass. The expression semantics are rich enough to allow complex regular expression matching and numeric evaluation but constrained enough to allow a KeyNote evaluator to be built into embedded applications or operation system kernels.

Our design philosophy for KeyNote thus departs from that of PolicyMaker, in which assertions can be arbitrary programs. PolicyMaker is designed to provide a general trust-management framework across a wide range of applications, at some expense of efficiency. KeyNote, on the other hand, provides somewhat simpler syntax and semantics, aimed specifically for building public-key infrastructure applications, at some expense of generality.

4 Extensibility

Two important components of a trust-management system are the assertion syntax and the compliance-checking algorithm. (Recall that "compliance checking" is the process of deciding whether a set of credential assertions prove that a request complies with a policy assertion.) In the PolicyMaker trust-management system [2, 3], both the assertion syntax and the compliance-checking algorithm are proper generalizations of the corresponding components of KeyNote. This implies that KeyNote is highly extensible; indeed, its natural extension has already been implemented. An application that needs more general assertions than KeyNote provides can easily upgrade its trust-management engine: It can switch from using the KeyNote compliance checker to using the PolicyMaker compliance checker, write its new, more general assertions in PolicyMaker, *and continue to use its old KeyNote assertions, because they are compatible with PolicyMaker assertions and can be processed by the PolicyMaker compliance checker.*

The PolicyMaker notion of "proof of compliance" is beyond the scope of this paper; it is discussed in full detail in [3]. Here we briefly explain one way in which PolicyMaker assertions generalize KeyNote assertions and why this generalization might be useful. A PolicyMaker assertion is a pair (f_i, s_i). The "source" s_i is basically identical to the KeyNote SIGNER field. The function f_i is a general program that may be written in any language that can be "safely" interpreted within the trust-management environment; in particular, f_i may be a KeyNote assertion. Rather than simply returning TRUE or FALSE, PolicyMaker assertions produce sets of "acceptance records." A record has the form (i, s_i, R_{ij}) and means that "assertion number i, issued by source s_i, approves action string R_{ij}." The action string *may* be the request that was fed to the trust-management system by the calling application, or it may be some related action-string that makes sense in the context of this attempted proof of compliance. Each time it is executed during an attempted proof, an assertion receives as input all of the acceptance records that have been approved so far by the policy assertion (f_0, POLICY) or by any of the credential assertions $(f_1, s_1), \ldots,$ (f_{n-1}, s_{n-1}). Such non-boolean assertions are useful in several standard trust-management constructions, including those that require explicit control over "delegation depth." See [3, Section 3] for more details.

Ellison *et al.* note that some applications may require more expressive power than the SPKI/SDSI certification framework provides, and they suggest the use of PolicyMaker to achieve this additional power. More precisely, [5, Section 7.3] contains the following suggestion: "For any trust policy which the full SPKI 5-tuple reduction can not express, one must write a policy interpretation program and PolicyMaker provides a language and body of examples for that purpose. The result of the PolicyMaker execution can be a 5-tuple to be used within an SPKI 5-tuple reduction." The observation that a PolicyMaker assertion may produce an acceptance record whose action string R_{ij} encodes a SPKI 5-tuple is a good one, and this may indeed be a reasonable way to extend SPKI. However, the precise relationship between the PolicyMaker definition of "proof of compliance" and the definition given by SPKI 5-tuple reduction has not yet been formally analyzed. (This is a potentially interesting direction for further research.) Thus we cannot formally characterize the additional power that this use of PolicyMaker would bring to the SPKI/SDSI framework or assess the ease with which such an extension could be implemented.

5 Conclusions

We have presented the design principles for KeyNote and contrasted our design with other existing proposals. We believe that the combination of simplicity, expressiveness, and extensibility makes KeyNote well-suited for trust-management in public-key infrastructure.

For more information about KeyNote, please read the Internet Draft [1].

References

1. M. Blaze, J. Feigenbaum, and A. D. Keromytis, "The KeyNote Trust Management System," work in progress. Internet Draft, April 1998, `http://www.cis.upenn.edu/~angelos/draft-angelos-spki-keynote.txt.gz`.
2. M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized Trust Management," in *Proceedings of the 17th Symposium on Security and Privacy*, IEEE Computer Society Press, Los Alamitos, 1996, pp. 164–173.
3. M. Blaze, J. Feigenbaum, and M. Strauss, "Compliance Checking in the PolicyMaker Trust Management System," in *Proceedings of the 2nd Financial Crypto Conference*, Lecture Notes in Computer Science, Springer, Berlin, 1998, to appear. Available in preprint form as AT&T Technical Report 98.3.2, `http://www.research.att.com/library/trs/TRs/98/98.3/98.3.2.body.ps`.
4. C. Ellison, *A Simple Public-Key Infrastructure*, `http://www.clark.net/pub/cme/html/spki.html`.
5. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI Certificate Theory," `http://www.clark.net/pub/cme/theory.txt`
6. R. Rivest and B. Lampson, *SDSI: A Simple Distributed Security Infrastructure*, `http://theory.lcs.mit.edu/~rivest/sdsi11.html`.
7. Consultation Committee, *X.509: The Directory Authentication Framework*, International Telephone and Telegraph, International Telecommunications Union, Geneva, 1989.