

Making certificates programmable

John DeTreville
 Microsoft Research
 johndetr@microsoft.com

Abstract

Certificates carry signed statements within a Public-Key Infrastructure (PKI). As we begin to build more complex and more open PKIs, the limited expressiveness of current certificate languages becomes a concern. While certificates are traditionally treated as simple data structures conforming to a given schema, we show an alternative derivation of the concept of a certificate in which certificates can contain control information in the form of program code. One example is program code written in declarative statements in a variant of the relational algebra, which can work together in rich ways.

1. Introduction

In a Public-Key Infrastructure (PKI)—such as X.509 [10] or SDSI/SPKI [13, 7]—distributed parties can communicate using persistent signed data structures called *certificates*. Certificates can carry authorizations that control access to distributed resources (saying, for example, that John Smith can access a particular Web site at his workplace) as well as more abstract data and rules that can provide support for authorization decisions (e.g., John Smith is a full-time programmer; programmers are employees; full-time employees can access the Web site). Certificates conform to an established syntax—such as ASN.1 for X.509 certificates [11] and encoded S-expressions or XML for SDSI/SPKI certificates [12]—and an established semantics.

As our ambitions for PKIs become greater, the expressiveness of their certificates can become a cause for concern. We might wonder whether our certificates—their syntax and their semantics—are expressive enough. Can they convey the necessary sorts of information to support the operation of the PKI? For example, if our certificates are very simple data structures that can work together only in a few restricted ways, it might be impossible to support a rich variety of authorization structures. While this may be seen as an advantage in some contexts (for example, if we might wish to constrain the uses of a PKI), it is certainly a potential shortcoming in a more open environment.

We might also wonder if our certificates and our certificate language are suitably well-defined. Ensuring the wide interoperability of certificates in an open PKI can be difficult or impossible in practice [9]. We note for example that certificates are often extended for new uses by simply adding new fields in a manner that can change the meaning of existing fields in subtle and perhaps unforeseen ways, breaking existing uses. Conversely, we might expect that a more regular design, based on fewer base concepts that can be used together in more ways, might improve interoperability while at the same time increasing expressiveness.

In this paper we rederive the concept of a certificate in a novel way, in which a certificate can contain program code, written in a simple declarative language, as well as data. The use of program code can increase the expressiveness of certificates while eliminating a number of special cases present in existing certificate languages, and

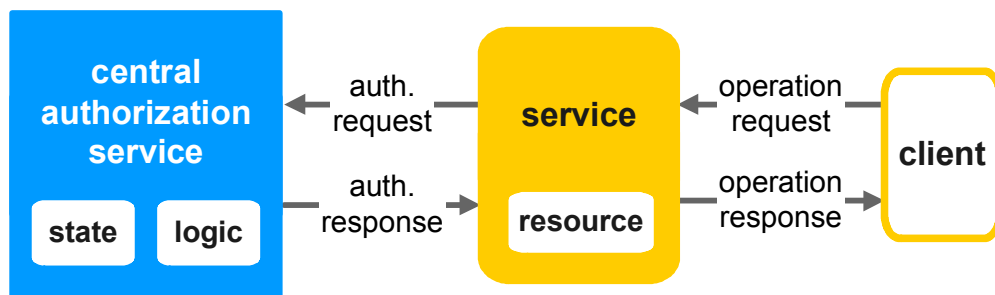


Figure 1: A hypothetical central authorization service

is one path toward deriving more powerful certificate languages that will allow us to build richer and more flexible PKIs.

2. A hypothetical central authorization service

The principal purpose of certificates—let us say—is to support authorization in an open distributed environment. Certificates therefore combine two distinct kinds of information. First, they include information directly related to authorization. For example, they may state that a certain group of people is authorized to access a shared resource, or that a certain person belongs to that group. They also include information required by their use in an open distributed environment. For example, they may include a validity interval, or an address to check for revocation, or information that supports the proper chaining of certificates.

To help separate these concerns, let us first consider a hypothetical environment where all authorization decisions have been centralized, as shown in Figure 1. Whenever a client requests an operation from a service controlling a resource, the service must determine whether this client is authorized to perform this operation; in this centralized model, the service simply passes an authorization request to the central authorization service, identifying the client, the resource, and the requested operation. Based on its encapsulated state and logic, the central authorization service authorizes or rejects the operation; if the operation is authorized, the service performs the requested operation and returns the result to the client. The central authorization service encapsulates the system's authorization information (its "state") and the authorization rules (its "logic") for all resources and for all clients, and it is used only as a "black box" that can only answer specific questions.

Such a centralized authorization service is of course impractical in many ways. Its performance and availability would be limited and it certainly could not scale to the size of the Internet. Worse yet, such a large-scale service would be impossible to administer, since it would combine information from thousands or millions of autonomous administrative domains and would hard-code the rules on how these domains operate and how they interoperate. It would be closed because third parties could not readily extend its state and logic.

Let us imagine, though, that our centralized distribution service is otherwise powerful enough to perform the needed authorization tasks, and that its only problems are those due to its centralized nature. How can we solve these problems, or at least ameliorate them? In other words, how can we decentralize (i.e., distribute) the authorization service?

3. Mobile code

One approach to decentralizing the authorization service is to make its state and logic *mobile*—that is, to encapsulate some piece of its state and logic in a certificate that can travel across the network to the service controlling the resource and execute there. There have been various proposals that support this sort of mobile code [4] and this approach is greatly simplified when the authorization process is purely functional—without side-effects—as is usually the case. We assume some mechanism for executing the code in the certificate safely at the receiving service.

Simply adding mobile code to our centralized design is not enough. It improves performance, and it improves availability, but it does not address the remaining problem of administering the system's global authorization state and logic. We can simply partition the state and logic, of course—and such a partition is clearly the solution—but the various partitioned administrative domains must still be able to interoperate. Below, we derive a architecture for partitioning that allows multiple administrative domains to interact in flexible ways. Our language for state and logic is purely applicative, thus allowing its safe execution at the recipient.

4. Certificates as cache entries

One way to improve performance and availability in any system is through the use of *caching*. Once a service sends a request to our hypothetical central authorization service and receives a response, it can cache the request-response pair to avoid requerying the central service for the same request in the future. Of course, the response must not depend on state that can change.

In their simplest form, certificates are an extension of the caching idea. As shown in Figure 2, a service can hold a certificate, signed by the central authorization service, encapsulating the request-response pair. It can use this certificate exactly as it would use the corresponding cache entry, but the certificate has several additional advantages.

- Cache entries are implicitly authenticated because the service (presumably) knows that the information in the cache came from the central authentication service, over an authenticated connection. In contrast, a certificate is explicitly authenticated because it carries a signature from the central authentication service. A service can trust a certificate received from another service, or even from a client. This feature further improves the performance and flexibility of the PKI.

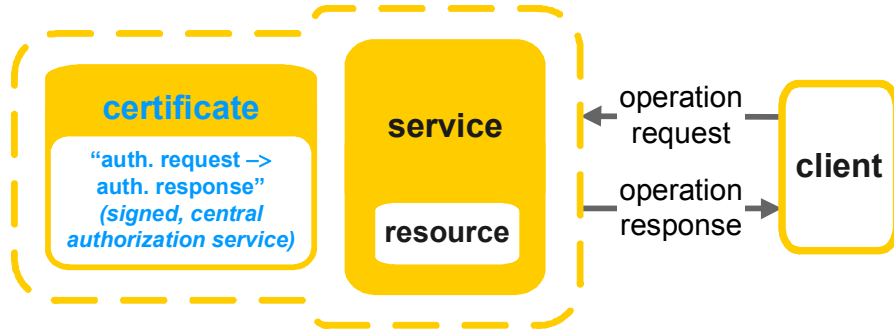


Figure 2: Certificate issued by a central authorization service

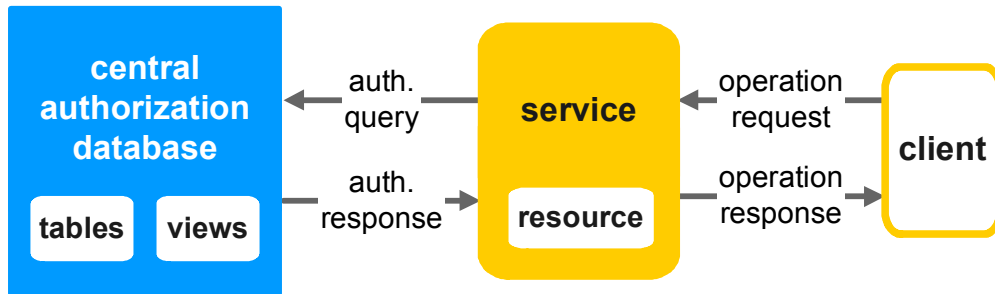


Figure 3: A hypothetical central authorization database

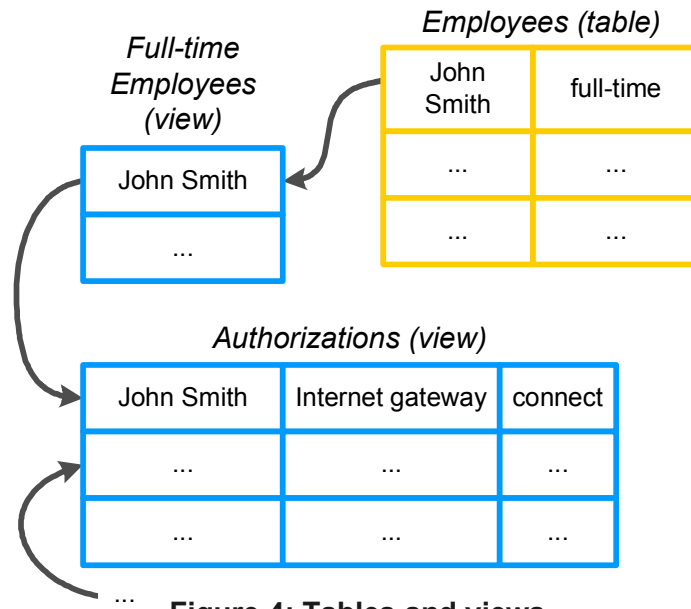


Figure 4: Tables and views in the central authorization database

- A certificate can potentially be obtained at a convenient time before it is needed. While a cache operates transparently, meaning that any request might need to contact the central authorization service, certificates allow us to explicitly collect—ahead of time—all of the information needed to authorize an operation, eliminating the need for the central authorization service to be available at the same time as each operation. This feature improves the availability of the PKI.
- Instead of supplying the response for one particular request, a certificate can contain wild cards, supplying the responses for a family of requests. For example, a certificate can say that a certain set of individuals—defined in some way—is authorized to perform a certain set of operations on a certain set of resources. This feature improves the performance and flexibility of the PKI. We will return to the idea of wild cards later in this paper.

In the simple use of certificates shown, the central authorization service remains a black box and does not expose or export its internal state and logic to its callers except in the form of request-response pairs. In the following sections we will make the black box more transparent by extending and regularizing the statements that certificates can carry.

5. Using a relational database to represent state and logic

To expose the internal structure of the central authorization service, it is necessary first to specify what forms the state and rules can take. In this section, we demonstrate how its state and logic can be modeled by a relational database [5, 8].

As shown in Figure 3, the central authorization database contains *tables* and *views*. Tables store data, while views are defined in terms of data that appear in tables and other views. The service receiving an operation request sends an authorization query to the database, and receives an authorization response.

Figure 4 shows the internal organization of one example database in further detail. Here, full-time employees are authorized to connect to an Internet gateway. An Employees table holds the names of the employees and their employment status. A Full-time Employees view is derived from the Employees table, and the final Authorizations view is further derived from the Full-time Employees view. In this simple example, the Employees table holds the raw data while the Full-time Employees view and the Authorizations view serve to encode the authorization logic.

When this example database is used, a service queries the Authorizations view at the authorization database, giving the client name (“John Smith,” or more generally a public key), resource name (“Internet gateway”), and operation name (“connect”) as keys. The database responds to the query by returning all matching rows. In this example, the database returns one row in case of authorization success, and zero rows in case of failure.

We can define the database views and queries in a number of forms, including relational algebra, which operates on tables and queries using operators like *select*, *project*, and *join*. In this paper, we extend the relational algebra with two additional operators.

- We add a *union* operator that combines tables or views with the same schema. Although the Authorizations view is shown here as a simple view on the Full-time Employees view, it would more generally be the union of a number of views, each of which might define authorizations on a particular resource, set of resources, etc.
- We also add *recursion*, to allow for the computation of transitive closures. This is useful for modeling authorization chains, as discussed below.

Because nonmonotonicity can be unsafe in a distributed environment, we additionally restrict our relational algebra to be monotonic by eliminating negation. It is a topic for future work to characterize those uses of nonmonotonicity and negation that nevertheless can be safely allowed.

While the schema of the Authorizations view must be partly standardized—and known to the services querying the authorization database—the schemas of the other views and tables need not be standardized at all. This can be seen as a significant advance over older PKI schemes like X.509 and even SDSI/SPKI. The tables can include arbitrary data with arbitrary structure, and the Authorizations view can be the result of arbitrary computations on these tables. (Of course, these computations must be expressible in our extended relational algebra; this is true for the classes of authorization problems that we have studied.)

Traditional security languages include special-case syntax and semantics for encoding extra conditions and information needed for authorization. Because of the use of arbitrary schemas and the power of the extended relational algebra, though, the authorization database can represent these conditions and information directly. For example, while SDSI/SPKI includes a mechanism for group membership, we note that our authorization database can model groups directly in the relational algebra, as in the example above. We can also represent different kinds of groups, such as groups of resources or groups of operations; this is impossible or limited in traditional languages. Similarly, we can model the idea of certification

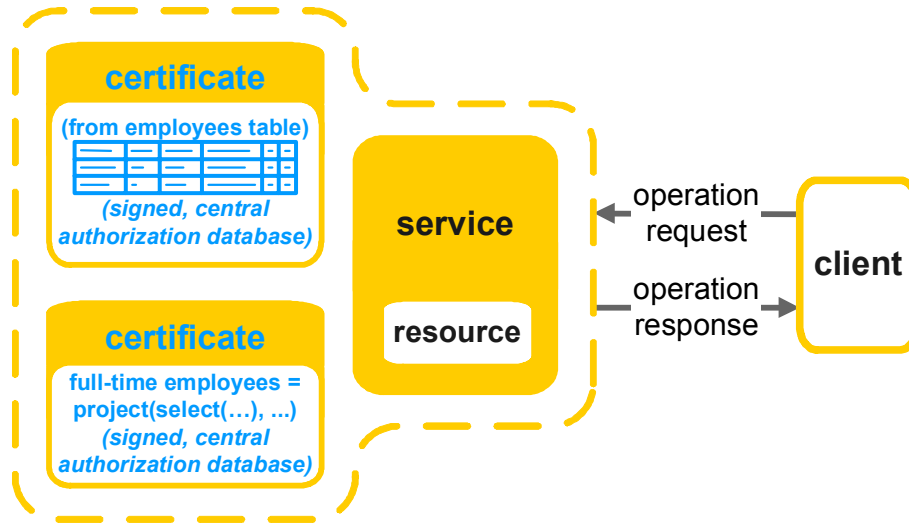


Figure 5: Certificates issued by a central authorization database

authorities and certificate chains, as in X.509 and SDSI/SPKI, directly in the extended relational algebra instead of building it into our language. (This requires the addition of recursion to the relational algebra, as discussed above.) Different administrative domains can be programmed to have different properties, and we can also generalize the use of one-dimensional chains to allow more complex and more general trust relations.

(We note that the relational algebra is closely related to the logic-programming language datalog [1]. The central authorization database can therefore be replaced by a program written in datalog or another logic-programming language, as in the Binder security language [6].)

Choosing to represent our authorization information and rules in a relational database system might seem as merely shifting our problems from one domain to another. However, there is a wealth of experience in designing good relational database schemas [2]—such as the use of normal forms—as well as formalizing the semantics of schemas. We believe that many of the problems of authorization are simplified by restatement in the context of databases, relational algebra, and logic programming. Furthermore, the greater generality of the database context can lead to a more general solution to the authorization problem.

6. Certificates as signed database excerpts

Certificates served to encapsulate request-reply pairs with our original central authorization service, and they play much the same role in conjunction with the central authorization database. However, since we can now expose some of the internal structure of the central authorization database—we can name its tables and its views and

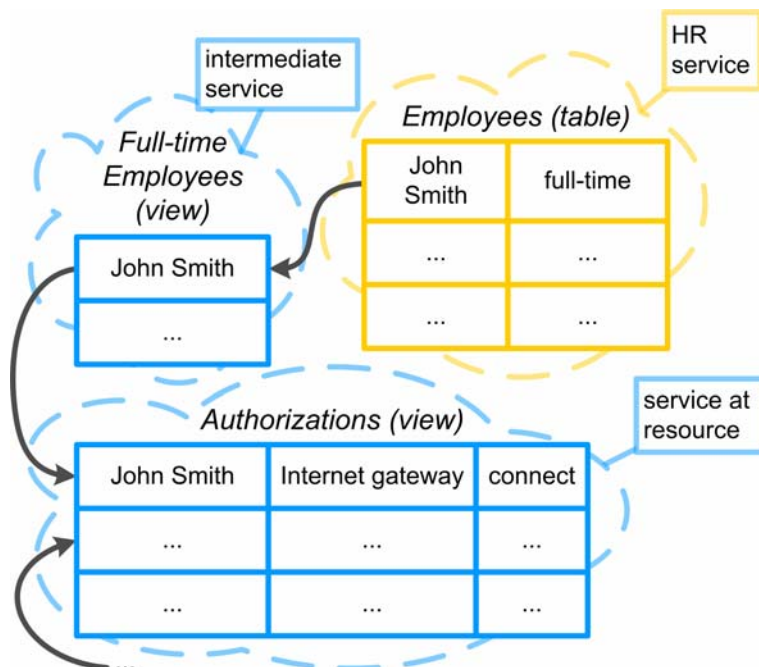
give their schemas and definitions—we can now store much richer information in our certificates.

As shown in Figure 5, services still use the certificates issued by the central authorization database in lieu of an on-line request and reply. Unlike the earlier use of certificates, though—in which certificates simply cached signed request-reply pairs—these certificates can store additional information which the services can use to derive future authorizations. Figure 5 outlines the two types of certificates that the central authorization database can now issue.

- The first type of certificate includes an excerpt—one or more rows—from a table or view. Here, the first certificate includes rows from the Employees table. This type of certificate states that the excerpted rows were found in the named table or view.
- The second type of certificate defines a view in terms of a relational algebra expression involving other tables and other views. Here, the second certificate includes the definition of the Full-time Employees view in terms of the Employees table.

These certificates are, of course, still signed by the central authorization database, and can be received from the central authorization database or from a client or other service. These database certificates name the table or view that their information comes from, and also include enough schema information to allow their interpretation at the service.

The database certificates can include enough information to derive the replies for many different requests. (This is an example of the wild-card feature described earlier.) Just as we do not require these certificates to include *all* of the rows of a table or view, they also need not



**Figure 6: Tables and views
in the distributed authorization database**

contain the *complete* definition of a view. For example, a database certificate encapsulating the Authorizations view—which might be the union of a large number of views—can simply say that it includes one particular view. Database certificates therefore contain only partial information; they can say only that a given authorization does exist, and cannot say that it does not. (Extensions to partially eliminate this restriction are possible but are outside the scope of this paper.)

Constraining the structure of the central authorization service to be a relational database thus allows our certificates to include richer, more general forms of information. Our central authorization database can issue certificates whose meaning cannot be represented in X.509 or in SDSI/SPKI—as illustrated below—and it regularizes the treatment of existing features.

7. Distributing the database

Our central authorization database is still centralized, and while the use of certificates has reduced the problems of performance and availability, they still exist. Worse, we have not attacked the administrative problems inherent in a centralized architecture. To eliminate these problems, we now show how to partition the central authorization database into a distributed authorization database.

Figure 6 illustrates the operation of the distributed authorization database. The database still contains tables and views, but they are stored in multiple services on the

network. In this example, for instance, a Human Resources (HR) service holds the Employees table, but the service controlling the resource itself can define the portion of the Authorizations view that it interprets. Yet another intermediate service can define the Full-time Employees view referenced by the Authorizations view.

Although most tables and views can be stored anywhere on the network, we require that the Authorizations view be distributed among the services that control the various network resources. The distributed authorization database thus follows the lead of the PolicyMaker language [3], in which the root of all authorization decisions is local by convention and is established administratively. Distributed certificates are still used in the same way as our earlier certificates. As shown earlier in Figure 5, a service controlling a resource can use multiple certificates to make authorization decisions. When these are distributed authorization certificates, they may come from multiple services.

As shown in Figure 7, certificates are signed by the services that issue them. Here, Employee certificates are signed by the HR service, while Full-time Employee certificates are signed by the intermediate service. The service at the resource need not sign its definition of the Authorizations view to use it, since it originates locally. Each definition of a view identifies the public key used by the tables or views it uses as inputs.

We have thus eliminated the need for the central database service to issue and sign certificates. Since multiple autonomous services can now issue certificates, we

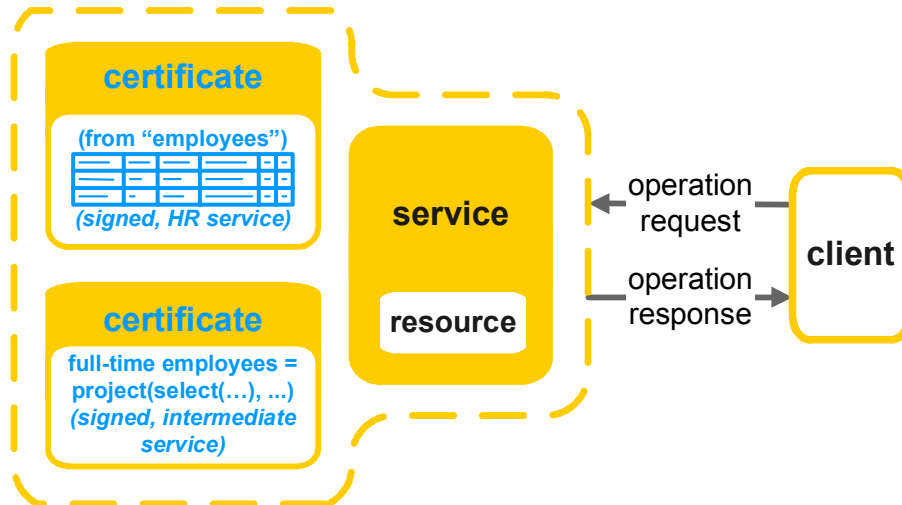


Figure 7:
Certificates issued by a distributed authorization database

can directly accommodate multiple administrative domains. Administrative domains can interoperate because they can explicitly refer to one another by the public keys of the issuing services. The resulting system is similar in many ways to traditional uses of certificates but it has some notable differences. In particular, references to public keys need not be constant, but can themselves be drawn from tables and views, as shown in Figure 8. Here, the policy expressed is that full-time employees can access the Internet gateway if authorized by their bosses. We combine our earlier Full-time Employees view with a Bosses view, as well as an Approvals view local to each boss.

Allowing views in one service to refer to tables or views in another allows the PKI designer to use an arbitrary number of levels of indirection. Since it is a folk theorem in Computer Science that any problem in computing can be solved by adding another level of indirection, we can expect that this will be a powerful technique, and that it will serve to make explicit and to extend some number of security assumptions that might otherwise be wired into the system architecture.

In particular, this distributed certificate structure provides a concrete interpretation of the abstract notion of “trust.” One service trusts another if its views depend on tables or views from that other service. Because the database can hold the names of services (e.g., their public keys), we can organize services into groups or other more complex relations. For example, we might have a table of which services “trust” which others. Certification Authorities are no longer special entities in our PKI; we can choose to implement them in the same form as in traditional PKIs—that is, their certificates can continue to bind names to identities, or to delegate the power to issue fur-

ther certificates—or we can choose different schemas that take advantage of our greater flexibility and generality.

9. Conclusions and future work

We have shown how certificates can be made more expressive and more precise by allowing them to include program code written in a language such as an enhanced relational algebra. While we have outlined the operation of such a system, much future work is clearly needed.

We have not touched on certificate revocation in this paper. While the standard techniques for revocation continue to apply, we would still like to understand how to make revocation programmable, as well as checking for revocation. More generally, we have assumed that the statements in our system has no side effects, which is clearly a poor assumption in many cases.

While making certificates programmable increases their expressiveness, greater expressiveness can always be misused and can in fact keep us from saying the right things by making it too easy to say the wrong things, or to understand the implications of our statements. Thus, the choice of a security language ultimately involves an engineering tradeoff between increasing generality and maintaining usability. Understanding this tradeoff again requires further experience.

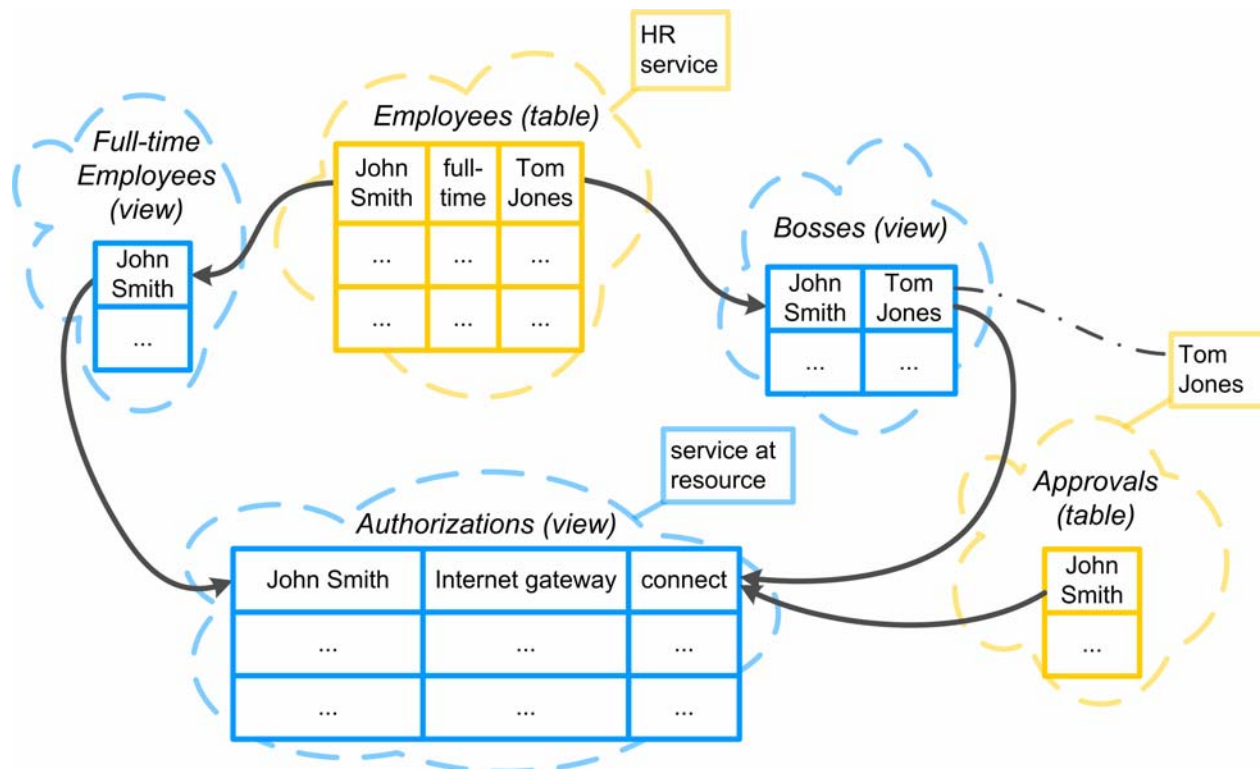


Figure 8: Adding another level of indirection

References

- [1] M. Ajtai and Y. Gurevich. "Datalog vs. first-order logic." In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 142–146, 1989.
- [2] J. Biskup. "Achievements of relational database schema design theory revisited." In B. Thalheim and L. Libkin, eds., *Semantics in Databases*, Lecture Notes in Computer Science, Vol. 1358, pages 29–54. Springer-Verlag, 1998.
- [3] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proc. 1996 IEEE Symp. on Security and Privacy*, May 1996.
- [4] L. Cardelli. "Abstractions for mobile computation." In J. Vitek and C. Jensen, eds., *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, vol. 1603 of LNCS, pp. 51–94. Springer-Verlag, 1999.
- [5] E. F. Codd. "A relational model for large shared data banks." *Comm. of the ACM*, 13(6):377–387, June 1970
- [6] DeTreville, J. 2002. "Binder: a logic-based security language." To appear, *Proc. 2002 IEEE Symp. on Security and Privacy*, May 2002.
- [7] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. "SPKI certificate theory." IETF RFC 1693, September 1999.
- [8] J. Gray, *et al.* "System R: Relational approach to database management." *ACM Trans. on Database Systems* 1(2), pages 97–137, June 1976.
- [9] P. Gutmann, "X.509 style guide," available at <http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt>, October 2000.
- [10] ITU-T Recommendation X.509, "The directory: public-key and attribute certificate frameworks." March 2000.
- [11] ITU-T Recommendation X.680. "Abstract Notation One (ASN.1): Specification of basic notation." December 1997.
- [12] X. Orri & Mas, J.M. 2001. "SPKI-XML certificate structure." IETF Internet Draft, November 2001.
- [13] R. Rivest and B. Lampson, "SDSI—a simple distributed security infrastructure," available at <http://theory.lcs.mit.edu/~cis/sdsi.html>.