A Global Authentication Service without Global Trust[1]

Andrew D. Birrell, Butler W. Lampson,
Roger M. Needham, and Michael D. Schroeder

Systems Research Center
Digital Equipment Corporation

Abstract

This paper describes a design for an authentication service for a very large scale, very long lifetime, distributed system. The paper introduces a methodology for describing authentication protocols that makes explicit the trust relationships amongst the participants. The authentication protocol is based on the primitive notion of composition of secure channels. The authentication model offered provides for the authentication of "roles", where a principal might exercise differing roles at differing times, whilst having only a single "identity". Roles are suitable for inclusion in access control lists. The naming of a role implies what entities are being trusted to authenticate the role. We provide a UID scheme that gives clients control over the time at which a name gets bound to a principal, thus controlling the effects of mutability of the name space.

Introduction

This paper describes a design for an authentication service for a distributed system. The design has three goals that we feel have not been met simultaneously by any previous design. First, the service must be able to grow to cover an arbitrarily large physical area, arbitrarily many administrative organizations, and arbitrarily many users (millions or billions); the service must be suitable for a long lifetime. Second, the system must not be monolithically trusted: it must be possible to achieve authentication even if there exist untrusted parts of the system. Third, these goals must be met in such a way that each party to the authentication knows precisely what agencies the party must trust in order to accept the authentication.

The fundamental purpose of authentication is to enable "principals" to identify each other in a way that allows them to communicate, with confidence that the communication originates with one principal and is destined for the other. The principals we are considering include people, machines, organizations and network resources such as printers, databases or file systems.

[1] This paper appeared in *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, April 1986, IEEE order number 716, pp 223-230.

All authentication schemes ultimately reduce to enabling each principal to obtain or possess some information identifying the other. In a distributed system, this information is then used as key to some encryption based security protocol. In this paper we will not discuss secure communication protocols—there is already an adequate literature on that topic [3,7].

The simplest solution to the authentication problem is for each pair of principals to possess a pairwise shared secret key; then authentication is achieved trivially using a two-way handshake to prove knowledge of the shared key [7]. Of course, in most large systems this is impractical: the problems of distributing or changing such keys are extreme, involving the use of N^2 secret keys for communication amongst N principals.

Another simple solution is available using public key encryption [5]. If each principal has the public key of each other principal, it is straightforward to establish secure communication. This requires only N keys, but it is still extremely difficult to change keys. In a large scale system containing millions or billions of principals, even the use of compact disc technology would not be sufficient for publishing the public keys, particularly considering how often keys would change.

All the remaining approaches, including the present design, are based on the use of a trusted intermediary known as a “key distribution center” or “authentication service”. In such a system it is necessary to be able to identify the principals. In some circumstances unique identifiers (such as 48-bit numbers) are sufficient for this, but in a large scale system the only practical approach is for the principals to be named by some form of distributed name service. In our view, a global authentication service is impractical unless it is based on some such global name service.

Note that if an authentication design is based on the notion of naming the principals through a network name service, then the design is relatively unperturbed by the decision to use a public key encryption system or a secret key one. Public key systems have the benefit that the name service database is less secret, but the flow of data and control amongst the principals is unaffected [5]. For the remainder of this paper, we will consider only secret key systems.

Background

The present authentication service design was developed as part of a design for a global name service. The goals of that design are similar to the ones stated above. The name service provides a heirarchic name space. The name space is formed from “directories”, each of which provides mutable mappings from a “simple name” (typically denoted by a character string) to a set of values. These directories are connected into a tree structure with a single, global, root, as shown in figure 1. The name space we provide is quite similar to the name space of many common file systems, such as Unix.

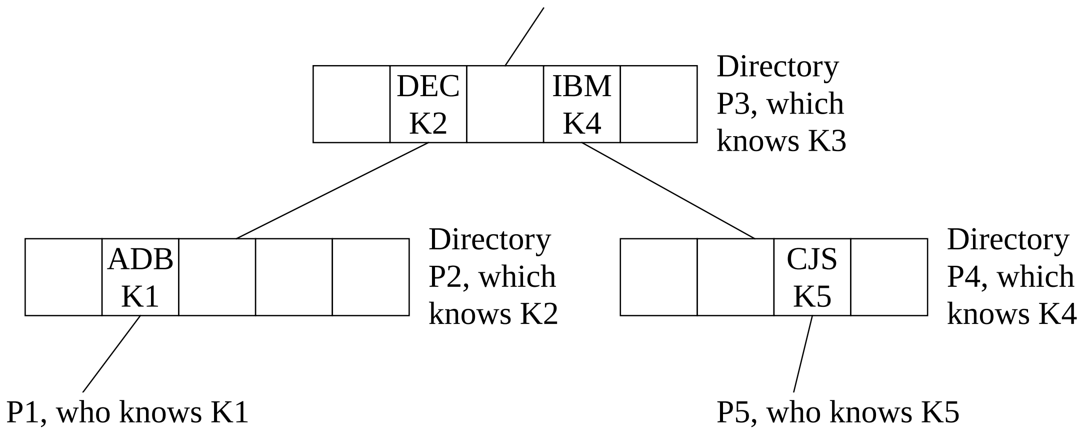
DEC
K2
IBM
K4
Directory P3, which knows K3
ADB
K1
Directory P2, which knows K2
CJS
K5
Directory P4, which knows K4
P1, who knows K1
P5, who knows K5

Figure 1: A corner of a hierarchic name space

Each directory may be replicated; each directory replica is maintained by a “name server” (running on some computer). Each name server maintains some set of (entire) directory replicas. The distribution and replication of the directories amongst the name server machines is similar in spirit to some earlier designs such as Grapevine [2] or Clearinghouse [8].

Each directory in the name service is permanently and uniquely identified by a UID. The “names” presented to this service consist of the identifier of some directory (known as the “root” of the name), and a path from there consisting of a sequence of simple names for the directories that form the path.

Some further features of the name space provided by this name service will be described later in the paper. A detailed description of this name service exists, although it is not yet in a form suitable for wide distribution (for pedagogic reasons, not completeness or correctness).

The easy way to use such a name service for authentication is to view the name service as uniformly trusted; then there are straightforward published authentication protocols [1, 4, 6, 7]. This level of trust is unacceptable for our present goals.

Authentication primitives

The basis for our authentication is a set of “secure channels” provided by the name service. A secure channel is formed whenever a pair of principals share a key. This is a secure channel in the sense that the two principals can readily use the key to establish communication encrypted by some conversation key, assured of freedom from eavesdropping, tampering or replay, and assured of the identity of the other principal.

Each principal registered in the name service has a secure channel to that principal’s directory. This is formed using the principal’s personal secret key (password), held by the principal and also stored in the principal’s directory entry. Implicit in our design is the notion that there is a one-to-one correspondence between principals and directory entries. Similarly, each directory has a secure channel to the directory's parent, formed by a secret key held by the directory and stored in the directory's entry in its parent directory. In this respect, a directory behaves just like any other principal.

We make extensive use of signed and sealed information. We write "$\{Q\}_K$" to mean that the message Q has been signed and sealed with the key K. The impact of this is that $\{Q\}_K$ can be manufactured only by a principal who possesses K, and given $\{Q\}_K$, Q can be extracted only by a principal who possesses K. (In a public key system, the operation of signing is performed with a principal's secret key, and sealing is performed with a principal's public key.)

We use the following notation for secure channels. If principals P_i and P_j each possess K_{ij} which forms a secure channel between them, we write "P_i knows $K_{ij} \rightarrow P_j$" and "P_j knows $K_{ij} \rightarrow P_i$". The symbol "$\rightarrow$" can be read as "authenticates". If P_i was given the channel by some other principal or set of principals P_*, then P_i should trust the channel only as much as P_i trusts P_*. We write this as "P_i knows P_* say $K_{ij} \rightarrow P_j$"

The impact of the "authenticates" notation is the following lemma:

a) If principal P_i knows	"P_* say $K_j \rightarrow P_j$"
and P_i receives	"$\{Q\}_{Kj}$"
then P_i may deduce	"P_* say P_j says Q".

Informally, (a) says that when P_i receives a message (Q) along a secure channel, then P_i should believe the message, subject to trusting the principal at the other end of the channnel (P_j) and the principal or principals who gave P_i the channel (P_*).

Our authentication schemes are based on this lemma and the following algorithm, known as the "forwarding rule":

b) If principal P_i knows	"$K_x \rightarrow X$" and "$K_y \rightarrow Y$"
and is asked to forward	"$\{P_*$ say $K \rightarrow P_j\}_{Kx}$" to Y,
then P_i deduces	"X says P_* say $K \rightarrow P_j$", using lemma (a),
and P_i replies	"$\{X$ says P_* say $K \rightarrow P_j\}_{Ky}$, $\{K \rightarrow Y\}_K$".

Here and everywhere else we are eliding the details required for secure communication along one of our secure channels. These details include challenge-response connection establishment using nonce identifiers, sequencing of requests and replies, and signing and sealing of all data with a conversation key. Any reader who is unable to fill in these details easily should first read a paper such as the survey by Voydock and Kent [7].

Informally, the value of the forwarding rule is that it allows us to compose secure channels to form a new secure channel. In particular, if we have a secure channel from P_1 to P_2 and another from P_2 to P_3, we can use the forwarding rule to construct a secure channel (that is, a shared secret key) from P_1 to P_3, subject to trusting P_2. This mechanism is applied three times in the following example.

Authenticating in a heirarchic name space

Consider a corner of a heirarchic name space as shown in figure 1. The labels P_1, P_2, etc. identify principals, and K_1, K_2, etc. are the corresponding secret keys. P_1 and P_5 are clients of the name service, while P_2, P_3 and P_4 are three directories. P_3 has some parent directory, not shown here. P_1 is known as “ADB” relative to P_2. P_2 is known as “DEC” relative to P_3. P_4 is known as “IBM” relative to P_3. P_5 is known as “CJS” relative to P_4. Each principal knows its own secret key, which is also known to its directory entry.

1.1:	P_1 knows:	$K_1 \rightarrow P_2$	
	P_2 knows:	$K_1 \rightarrow P_1$ and $K_2 \rightarrow P_3$	
	P_3 knows:	$K_2 \rightarrow P_2$ and $K_4 \rightarrow P_4$ and $K_3 \rightarrow P_3$’s parent	
	P_4 knows:	$K_4 \rightarrow P_3$ and $K_5 \rightarrow P_5$	
	P_5 knows:	$K_5 \rightarrow P_4$	
	P_1 creates a new key K and decides to use it such that $K \rightarrow P_2$		
1.2:	P_1 sends P_2:	please forward $\{K \rightarrow P_1\}_{K1}$ to P_3	
	P_2 replies:	$\{P_1$ says $K \rightarrow P_1\}_{K2}$, $\{K \rightarrow P_3\}_K$	using (b)
	P_1 deduces:	P_2 says $K \rightarrow P_3$	using (a)
1.3:	P_1 sends P_3:	please forward $\{P_1$ says $K \rightarrow P_1\}_{K2}$ to P_4	
	P_3 replies:	$\{P_2$ says P_1 says $K \rightarrow P_1\}_{K4}$, $\{K \rightarrow P_4\}_K$	using (b)
	P_1 deduces:	P_2 says P_3 says $K \rightarrow P_4$	using (a)
1.4:	P_1 sends P_4:	please forward $\{P_2$ says P_1 says $K \rightarrow P_1\}_{K4}$ to P_5	
	P_4 replies:	$\{P_3$ says P_2 says P_1 says $K \rightarrow P_1\}_{K5}$, $\{K \rightarrow P_5\}_K$	
	P_1 deduces:	P_2 says P_3 says P_4 says $K \rightarrow P_5$	using (a)
1.5:	P_1 sends P_5:	$\{P_3$ says P_2 says P_1 says $K \rightarrow P_1\}_{K5}$	
	P_5 deduces:	P_4 says P_3 says P_2 says P_1 says $K \rightarrow P_1$	using (a)

Figure 2: Establishing a secure channel between P_1 and P_5.

Figure 2 shows the steps involved in establishing a secure channel between P_1 and P_5. The final state is that P_1 and P_5 share K, and each knows that K authenticates the other, subject to trusting the intermediate principals.

This is all fine, except for the question of how we identify and represent the principals being discussed. If we identify them using a global, absolute, name, then we are effectively trusting the higher layers of the naming hierarchy all the way back to the global root. This is unacceptable for the sort of system we are designing. This problem is resolved by observing that the principals being trusted by P_1 are precisely those that form a path from P_1 to P_5, and that P_5 trusts those that form a path from P_5 back to P_1. We can take advantage of this by applying the following rewriting rule:

c) The statement "A says $K \rightarrow B$", with B named relative to A,
is equivalent to "$K \rightarrow A/B$".

Conversely, a statement of the form "$K \rightarrow A/B$" should be interpreted as implying that A says $K \rightarrow B$. In other words, that K should be trusted as authenticating B, relative to A, to the extent that A is trusted.

We can use (c) to restate a special case of lemma (a) as follows:

d) If principal P_i knows "$K_j \rightarrow P_j$"
and P_i receives "$\{K \rightarrow P\}_{Kj}$"
then P_i may deduce "$K \rightarrow P_j/P$".

Similarly, we can restate the forwarding rule (b) as:

e) If principal P_i knows "$K_x \rightarrow X$" and "$K_y \rightarrow Y$", named relative to P_i,
and P_i is asked forward "$\{K \rightarrow P_j\}_{Kx}$" to Y,
then P_i deduces "$K \rightarrow X/P_j$", using lemma (d),
and replies "$\{K \rightarrow X/P_j\}$, $\{K \rightarrow Y\}_K$".

Now consider figure 3, which shows the same example, establishing a secure connection between P_1 and P_5, but reworked using relative paths. We use the notation ".." to mean "parent" and "." to mean "self". We replace occurrences of "P/." with "P" without comment.

The deductions being made by P_1 in step 2.4 and by P_5 in step 2.5 are exactly equivalent to the deductions they made in steps 1.4 and 1.5, except for the translation to relative paths. However, these paths are now in a form that is independent of other, untrusted parts of the name space.

These authenticated named paths are suitable for use in making access control decisions. For example, a file server (acting as a principal) would build its access control lists to contain paths relative to itself. When the file server authenticates a client, it obtains a path (just as P_1 did in 2.4) and then compares this path with the access control list for the requested file operation. Access control lists can also contain patterns matching paths, for example to give access to all principals in some organizational directory.

2.1:	P_1 knows:	$K_1 \rightarrow ..$	
	P_2 knows:	$K_1 \rightarrow$ADB and $K_2 \rightarrow ..$	
	P_3 knows:	$K_2 \rightarrow$DEC and $K_4 \rightarrow$IBM and $K_3 \rightarrow ..$	
	P_4 knows:	$K_4 \rightarrow ..$ and $K_5 \rightarrow$CJS	
	P_5 knows:	$K_5 \rightarrow ..$	
	P_1 creates a new key K and decides to use it such that $K \rightarrow ..$		
2.2:	P_1 sends P_2:	please forward $\{K \rightarrow .\}_{K1}$ to ..	
	P_2 replies:	$\{K \rightarrow \text{ADB}\}_{K2}$, $\{K \rightarrow ..\}_K$	using (e)
	P_1 deduces:	$K \rightarrow$../..	using (d)
2.3:	P_1 sends P_3:	please forward $\{K \rightarrow \text{ADB}\}_{K2}$ to IBM	
	P_3 replies:	$\{K \rightarrow \text{DEC/ADB}\}_{K4}$, $\{K \rightarrow \text{IBM}\}_K$	using (e)
	P_1 deduces:	$K \rightarrow$../../IBM	using (d)
2.4:	P_1 sends P_4:	please forward $\{K \rightarrow \text{DEC/ADB}\}_{K4}$ to CJS	
	P_4 replies:	$\{K \rightarrow \text{../DEC/ADB}\}_{K5}$, $\{K \rightarrow \text{CJS}\}_K$	using (e)
	P_1 deduces:	$K \rightarrow$../../IBM/CJS	using (d)
2.5:	P_1 sends P_5:	$\{K \rightarrow \text{../DEC/ADB}\}_{K5}$	
	P_5 deduces:	$K \rightarrow$../../DEC/ADB	using (d)

Figure 3: The same example reworked using relative paths.

Non-heirarchic naming: symbolic links

Our name service also provides facilities for non-heirarchic naming. This seems to be essential for the scale of system we envisage, and for any name service having a long lifetime. The primitive facility is that a directory entry may be marked as a “link” to another name. When such an entry is encountered during name resolution, the value of the entry is prepended to the remainder of the path being resolved.

For example, assume there is an entry named “Friend” in the directory P_2 of the previous example. The entry “Friend” is marked as a link, and its value is a name that resolves to the directory P_4. Then the path “Friend/CJS” from directory P_2 resolves to the directory entry for P_5. Similarly, the path “DEC/Friend/CJS” from the directory P_3 resolves to the same entry.

By treating symbolic links as additional secure channels, we can use them to bridge untrusted areas of the name space. We can make them secure by recording a key “K_f” in the entry “Friend” in directory P_2, and recording the key K_f in the private data structures of directory P_4. Note that if we do not trust P_3, then we must install this key in those places by external means—we cannot use P_3 to establish a channel from P_2 to P_4 for key installation. This is fundamental and unavoidable. Note also that our authentication will not rely on the accuracy of resolving the name given by the “Friend” directory entry, because we can verify that the name is correctly resolved by the fact that P_4 possesses K_f. Thus the name for “Friend” can happily involve untrusted regions of the name space, even up to the global root.

With this symbolic link, consider the alternative authentication from P_1 to P_5, as shown in figure 4.

3.1:	P_1 knows:	$K_1\rightarrow ..$	
	P_2 knows:	$K_1\rightarrow ADB$ and $K_2\rightarrow ..$ and $K_f\rightarrow Friend$	
	P_4 knows:	$K_5\rightarrow CJS$ and possesses K_f	
	P_5 knows:	$K_5\rightarrow ..$	
	P_1 creates a new key K and decides to use it such that $K\rightarrow ..$		
3.2:	P_1 sends P_2:	please forward $\{K\rightarrow .\}_{K1}$ to Friend	
	P_2 replies:	$\{K\rightarrow ADB\}_{Kf}$, $\{K\rightarrow Friend\}_K$	using (e)
	P_1 deduces:	$K\rightarrow ../Friend$	using (d)
3.3:	P_1 sends P_4:	please forward $\{K\rightarrow ADB\}_{Kf}$ to CJS	
	P_4 replies:	$\{K\rightarrow \mathbf{?}/ADB\}_{K5}$, $\{K\rightarrow CJS\}_K$	using (e)
	P_1 deduces:	$K\rightarrow ../Friend/CJS$	using (d)
3.4:	P_1 sends P_5:	$\{K\rightarrow \mathbf{?}/ADB\}_{K5}$	
	P_5 deduces:	$K\rightarrow ../\mathbf{?}/ADB$	using (d)

Figure 4: Authentication involving a symbolic link

In one direction, this is quite satisfactory. In step 3.3, P_1 deduces the desired authentication information. P_1 can now make access control decisions based on this path. P_5’s deduction in step 3.4 is less satisfactory, as the “?” symbols indicate. This comes from the fact that P_4 has no way of naming the incoming symbolic link, even although it possesses and can find K_f.

This asymmetry comes directly from the asymmetry of our name space. Not all paths through it are reversible. There are various ways to handle this problem. The simplest is to live with it. This would mean that the authentication service provides two-way authentication if and only if the path is reversible. An alternative would be to change the name space so that symbolic links are always bi-directional. Then the “?” in the above example would be replaced by the reverse name for the link, and both principals would derive satisfactory conclusions. In the present state of our design, we are not sure which of these to adopt.

In some cases, this asymmetry is in fact desirable. For example, if a manager wished to log in to a database server, the manager could be authenticated in his role using a symbolic link from a “manager” directory entry to his personal directory entry, but the manager would name the database server with the name of its ordinary identity.

Semantics

The authentication facility provided by this service may be unfamiliar. When P_5 receives an authenticator from P_1, then P_5’s knowledge of P_1’s identity consists of the path from P_5 to P_1. This contrasts with schemes where P_5 ends up knowing an absolute name for P_1.

Our view is that this authentication scheme authenticates “roles”, not “identities”. A principal may take on differing roles, depending what privilege he wants to exercise. For example, a user might normally use his role as an unprivileged programmer, but at other times exercise a role as a system administrator or a different role as a contracted consultant to an outside corporation. This is different from schemes where a user logs in with a different identity for each role; in our scheme the user retains the same identity (as evidenced by a particular object in a particular directory) regardless of his current role.

Thus, in example 2, P_5 is exercising the role “../../IBM/CJS”. In example 3 (with the symbolic link), P_1 is authenticating P_5’s use of the role “../Friend/CJS”.

An access control facility, for example, would maintain access control lists giving appropriate access rights to each role, or to patterns that match sets of related roles.

A significant advantage of the notion of “role” is that it allows a principal to decide what privileges to exercise at what times. This contrasts with other schemes that give a principal at all times whatever privileges the access control policies specify.

Although a principal exercises different roles, the principal uses just one password, the secret key corresponding to his identity. This contrasts with other schemes where a principal must present an additional password to exercise additional privileges, with the consequent problems of remembering and changing multiple passwords.

In considering the trust relationships that a principal believes when accepting an authenticated role, note that each principal trusts precisely the directories that form the nodes on the path described by the role. Thus placing a role in an access control list implies trust of precisely these nodes, but of no other principal.

Mutability of the name space

Our name service design includes the ability to rename a directory. This includes the ability to give a directory (and its entire descendent sub-tree) a new parent. It is also permissible to change the targeting of a symbolic link.

This mutability has substantial impact on the use of roles for access control decisions. Changing the arcs in its path might change the “meaning” of a particular role. There are occasions when this is desirable, and the intended access is for whatever entity, dynamically, is named by the path. On other occasions, the intended access is for the entity that was named by the path when the access right was given.

We give clients of the service control over this choice. Each directory is permanently and uniquely identified (for example, by a 96-bit UID). When an arc in a path names a directory, the arc may include the directory's UID. The effect is that if all the arcs in a path contain UID's, the path is totally bound to the particular directories. If only the end point of a path has a UID, it identifies a particular principal independently of how that principal is currently approached. A path with no UID's identifies whatever principal the path presently resolves to. In effect, this choice gives the client control over a spectrum ranging from extreme early binding to extreme late binding.

Summary and conclusions

We have presented a new authentication scheme that remains workable even in a very large scale, very long lifetime, distributed system. Our scheme ensures that each party to the authentication knows who is being trusted, and that it is possible to achieve two-way authentication without trusting the entire global authentication service. Our authentication semantics are based on the notion of "role", which is an authenticated path through the name space. Roles are suitable for inclusion in access control lists. Implicit with a role is the set of entities that must be trusted to authenticate the role. Use of differing roles allows principals to choose what privilege they wish to exercise.

The optional use of UID's in roles gives each access control agency control over the choice between early and late binding of names.

We believe that our design exemplifies the use of simple but powerful reasoning about authentication protocols. It is clear at all stages who knows what, and who is trusting whom. The "forwarding" rule is an important primitive. By composing secure channels, we can design authentication protocols with various desired properties. For example, we could compose the secure channel from P_2 to P_3 with that from P_3 to P_4 to obtain a cached channel from P_2 to P_4. (Such caching would be necessary in an implementation to obtain satisfactory performance.)

The design is at an advanced stage, and is specified fully by a semi-formal abstract notation, as is the global name server design. At this date, we have not implemented the design, but we intend to do so.

References

1. Bauer, R. K., Berson, T. A., and Feiertag, R. J. A key distribution protocol using event markers. *ACM Trans. Computer Syst*ems **1**, 3 (August 1983), 249-255.

2. Birrell, A. D., Levin, R., Needham, R.M. and Schroeder, M. D. Grapevine: An exercise in distributed computing. *Comm. ACM* 25, 4 (Apr. 1982), 260-274.

3. Denning, D. E. R. *Cryptography and Data Security*. Addison-Wesley, Reading Mass., 1982.

4. Denning, D. E., and Sacco, G. M. Timestamps in key distribution protocols. *Comm. ACM* **24**, 8 (Aug. 1981), 533-536.

5. Kline, C. S. and Popek, G. J. Public key vs. conventional key encryption. *AFIPS Conference Proceedings* **48**. AFIPS Press, Arlington, Va., 1979, 831-837.

6. Needham, R. M., and Schroeder, M. D. Using encryption for authentication in large networks of computers. *Comm. ACM* **21**, 12 (Dec. 1978), 993-999.

7. Voydock, V. L. and Kent, S. T. Security mechanisms in high-level network protocols. *ACM Computing Surveys* **15**, 2 (Jun. 1983), 135-171.

8. Xerox Corporation. Clearinghouse Protocol, XSIS 078404. Stamford, Connecticut, 1984.