



REPORT

Computer Science Technical Report: 96-24

November 1995

Some Hints on the Theory and Practice of Authentication in Distributed Systems

Tage Stabell-Kulø

INSTITUTE OF MATHEMATICAL AND PHYSICAL SCIENCES

Department of Computer Science

University of Tromsø, N-9037 TROMSØ, Norway, Telephone +47 77 64 40 41, Telefax +47 77 64 45 80

Abstract

Authentication in Distributed Systems: Theory and Practice [Lampson92] provides considerable insight. However, it can be hard to read, as many details are left out, probably for brevity; it is still 45 pages long. We provide detailed explanations of tricky points.

Introduction

This document serves two purposes. First, it was written to ensure that we understand [Lampson92], by providing omitted calculations and arguments. Second, by making it available we hope that others may struggle less than we did. Since this is a paper explaining elements in another paper, we do not claim that any novel aspects can be found herein.

On notation: “The paper” or “this paper” or “that paper” implies [Lampson92], and all references, such as “as described in section 7”, refer to the relevant section in the paper. This also holds for figures and tables, where appropriate. Particularly, $(PN, 1 \leq N \leq 12)$ refers to one of the 12 axioms and theorems about principals in section 3.2.

We suggest that students with a desire to learn about security in general and authentication in particular, find a different starting point than this paper. This is so as we believe that a good grasp on the inherent complexity of authentication is required in order to fully enjoy the issues considered in this paper.

A better place to start would be [Needham93]. Then focus on authentication by means of [BAN90], including the appendix [Burrows94]. Follow up with [Abadi94] and [Gong90]. A survey of authentication in distributed systems can be found in [Liebl93]. A tutorial in security at large is available in [Stallings95].

Channels and Encryption

Encryption Channels

There is more to key identifiers than meet the eye. The most important issue is that it is impossible to know whether a message has been correctly decrypted unless one knows the contents in advance. “Decrypted correctly” implies one of two: That the correct key has been used, and/or that the message was unaltered. When a human is the intended recipient, this can be achieved by including some (human) readable text¹ in the message.

When the message contains random bits, like a new key, some care must be taken to ensure that the recipient can verify that the message has been decrypted correctly, and that the correct key was used. This is not necessary if the protocol is guaranteed to detect this fact later.

Principals with names

A single Certification Authority

This section describes the use of an on line agent O , as put forward in section 5.1. We have an highly secure certification authority CA with the key K_{ca} . As long as CA is on line, there are no difficulties, since it can issue CA **says** $K_a \Rightarrow A$ whenever some principal needs to speak with A ; assuming it can be established that $K_{ca} \Rightarrow CA$.

¹“If you can read this, the message has been decrypted correctly”. However, a fixed text should not be used since one is then exposed for “known text” attacks.

There are two disadvantages related to the certificate K_{ca} **says** $K_a \Rightarrow A$. The first is that it can not be revoked unless it times out. This again makes it necessary to either give it a very long time to live, or having CA on line to reissue it whenever it times out.

Keeping a key secret is less hard when the key is off line² and we want a scheme where we have an on line agent that can make certificates previously made by CA valid.

In other words, the certificates made by CA is not valid unless O , the on line agent, verifies them. But O alone can not make anything valid that CA has not already certified.

Instead of certifying that K_{ca} **says** $K_a \Rightarrow A$, CA makes the weaker certificate

$$K_{ca} \text{ says } (O|K_a \wedge K_a) \Rightarrow A \quad (1)$$

O then counter signs this by issuing

$$O|K_a \text{ says } K_a \Rightarrow O|K_a \quad (2)$$

From these two, $K_{ca} \Rightarrow A$, and (P12) we again get $K_a \Rightarrow A$ if we proceed as follows:

We start with (1): In prose, it is:

“ K_{ca} assert that $K_a \Rightarrow A$, if O confirms it”.

We assume that $K_{ca} \Rightarrow A$, and with (1) and (P8) yields

$$A \text{ says } (O|K_a \wedge K_a) \Rightarrow A$$

(P10) is used to obtain

$$O|K_a \wedge K_a \Rightarrow A \quad (3)$$

In prose, (2) is: “ O quoting the key K_a implies (**says**) that K_a alone is as good as having O saying that it is good”. Or, in other words, K_a is still valid. Now, (2) with (P10) gives

$$K_a \Rightarrow O|K_a \quad (4)$$

The meaning of (4) becomes less obscure if we use (P7) and substitute K_a for A and $O|K_a$ for B :

$$(K_a \Rightarrow O|K_a) \equiv (K_a = K_a \wedge O|K_a)$$

using K_a by itself is as good as having O saying it. This certificate has a limited validity (in time).

We now substitute (4) in (3) and get

$$K_a \wedge K_a \Rightarrow A$$

which, by (P4), gives us the desired

$$K_a \Rightarrow A.$$

²Stored in a safe deposit box, for example.

Path Names and Multiple Authorities

It is not easy to obtain the public keys of others in a secure way. There are two main reasons why a centralized database is not a good solution. First, it will be heavily loaded. Second, everyone will have to trust it; finding an organization or person that everyone trusts is hard.

Even though we want to arrange certification authorities in a tree, we do not want to give “/” the authority to speak for everyone. For example, if we have the following two certificates

$$\text{/dec says /} \Rightarrow \text{/dec} \quad (5)$$

so that we can believe that

$$K_{\text{/dec}} \Rightarrow \text{/dec}$$

when

$$\text{/ says } K_{\text{/dec}} \Rightarrow \text{/dec}$$

and

$$\text{burrows says /dec} \Rightarrow \text{burrows} \quad (6)$$

so that we can believe that

$$K_{\text{burrows}} \Rightarrow \text{burrows}$$

when

$$\text{/dec says } K_{\text{burrows}} \Rightarrow \text{burrows}$$

we end up with

$$\text{/} \Rightarrow \text{burrows}$$

since \Rightarrow is transitive; **burrows** might find this not to be true. Furthermore, how is **burrows** going to obtain the public key of **root**³ unless by obtaining the three certificates

$$\text{root says } K_{\text{root}} \Rightarrow \text{root}$$

and

$$K_{\text{dec}} \text{ says } K_{\text{root}} \Rightarrow \text{root}$$

and

$$K_{\text{root}} \text{ says dec} \Rightarrow \text{root}$$

in order to use (P8) and deduce that

$$K_{\text{root}} \Rightarrow \text{root}$$

The problem is that **root** might find it somewhat inconvenient to issue the last of those certificates.

We need a mechanism enabling us to build a tree, where trust is localized but (nevertheless) transitive. The underlying problem is twofold: handoff is unconditional and “ \Rightarrow ” is transitive. And as we have seen, the problem is identical upwards and downwards in the tree.

So instead of handing off to **root**, **dec** will acknowledge that the channel C_{root} indeed speaks for **root**. This is an “upward” pointing certificate.

dec must issue two certificates. One, pointing down towards **burrows**. The other, pointing up towards **/**.

We have a number of certificates and by using them and the assumption

$$C_{\text{burrows}} \Rightarrow \text{/dec/burrows except nil}$$

³Or the key to any node above the node above himself.

we deduce in turn the body of each certificate, because for each A' **says** $C' \Rightarrow B'$ we thus can apply (P11) to get $C' \Rightarrow B'$.

We start out with the first certificate:

$$C_{burrows}|'...' \text{ says } C_{dec} \Rightarrow /dec \text{ except burrows}$$

Together with the assumption and (P8) we get

$$/dec/burrows|'...' \text{ says } C_{dec} \Rightarrow /dec \text{ except burrows}$$

we then use (N3) and obtain

$$/dec \text{ except burrows says } C_{dec} \Rightarrow dec \text{ except burrows}$$

and by using (P10) we get the desired

$$C_{dec} \Rightarrow dec \text{ except burrows}$$

which is what *Burrows* needs to trust the channel from '...'.
This shows how to obtain the necessary proof for a channel to a node higher in the tree. After reaching a common ancestor, the path is followed downwards in a similar way.

Roles and Programs

Loading Programs

If A doesn't trust the file system, it computes the digest D of the program text and looks up the name P to get credentials for $D \Rightarrow P$. Obtaining the certificate in a trusted way takes away the need for trusting the file system.

Not trusting the underlying file system creates a difficult booting problem, which is discussed in section 6.2. In general, however, most file systems are not local to the machine, but rather mounted across some network.

Booting

A secure system consists of a chain of certificates linking two principals together. The security of any system is no stronger than that of the weakest link. It is indeed an interesting problem to devise a practical installation procedure. Allowing users to have physical access to a machine which contains a secret key might turn out to be impossible. We are then faced with a setting where personal computing is no longer possible. Or, more likely, personal machines over which one has physical control will replace the typical workstation of today.

Delegation

When handing off its authority to another principal, e.g. A **says** $B \Rightarrow A$ there are two issues to consider:

- The handoff is unconditional. And if a certificate that hands off authority needs to be verified (in time) an online certification service must be added; as discussed in section 5.1.
- When A hands off its authority to B there is no difference between A and B since, by (P7), we have that $(B \Rightarrow A) \equiv (B = B \wedge A)$. So, if we want A to acknowledge what B says, we need delegation rather than handoff.

When B has been given the right to speak for A by A we end up with the conclusion that $B \Rightarrow A$ even though B might not know that this is the case. If B is careless, A might suffer; how is B to know that it speaks for A ? Delegation is less strict than handoff in that a delegation must be acknowledged.

Given the axiom

$$A \wedge B|A \Rightarrow B \text{ for } A \quad (D1)$$

and the two certificates

$$A \text{ says } B|A \Rightarrow B \text{ for } A \quad (7)$$

and

$$B|A \text{ says } B|A \Rightarrow B \text{ for } A \quad (8)$$

we obtain

$$(A \wedge B|A) \text{ says } B|A \Rightarrow B \text{ for } A$$

and by (D1) we get

$$B \text{ for } A \text{ says } B|A \Rightarrow B \text{ for } A$$

which by (P11) yields

$$B|A \Rightarrow B \text{ for } A$$

The important issue becomes visible in the conclusion, since only when B explicitly quotes A does it speak for $B \text{ for } A$. This is essential during login, where the user delegates to the workstation some authority, and the workstation accepts it. Note that by accepting the delegation the workstation has acknowledged that the user is logged on.

At login, a session key K_l is constructed. The user delegates to the combination of the session key and the workstation by issuing

$$K_u \text{ says } (K_w \wedge K_l)|K_u \Rightarrow K_w \text{ for } K_u$$

which in prose is:

The user, represented by the key K_u says that when the workstation, represented by the key (K_w), and the session key (K_l) *together* quote the user, they speak for the workstation **for** the user.

The user has tied together the three principals: the user, the workstation and the session key, or in other words, the user delegates some authority to *this* workstation (K_w) during *this* login session (K_l).

The workstation now accepts the delegation, and acknowledges that it must cooperate with the session key:

$$K_w|K_u \text{ says } (K_w \wedge K_l)|K_u \Rightarrow K_w \text{ for } K_u$$

and with the session key does the dual

$$K_l|K_u \text{ says } (K_w \wedge K_l)|K_u \Rightarrow K_w \text{ for } K_u$$

The situation is now that both the workstation and the login key has acknowledged each other. With the use of (P1) we get

$$(K_w|K_u \wedge K_l|K_u) \text{ says } (K_w \wedge K_l)|K_u \Rightarrow K_w \text{ for } K_u$$

Since $|$ distributes over \wedge this can be written as

$$(K_w \wedge K_l)|K_u \text{ says } (K_w \wedge K_l)|K_u \Rightarrow K_w \text{ for } K_u$$

which by (P10) leads us to

$$(K_w \wedge K_l) | K_u \Rightarrow K_w \text{ for } K_u \quad (9)$$

However, it is cumbersome for the workstation to countersign all communication with K_l . To remedy this, K_l signs a short term certificate

$$K_l \text{ says } K_w \Rightarrow K_l$$

which by (P10) ensures that $K_w \Rightarrow K_l$. Since $|$ is monotonic and distributes over \Rightarrow , as described in section 3.2, $K_w \Rightarrow K_l$ implies that

$$K_w | K_u \Rightarrow K_l | K_u \quad (10)$$

Since $|$ distributes over \wedge we can write (9) as

$$K_w | K_u \wedge K_l | K_u \Rightarrow K_w \text{ for } K_u$$

If we now use (10) and (P12) we get the desired

$$K_w | K_u \Rightarrow K_w \text{ for } K_u$$

Note that this is an optimization in that K_l has handed off all its authority to K_w and this is why the certificate $K_l \text{ says } K_w \Rightarrow K_l$ must be given a fairly short lifetime.

Authenticating Interprocess Communication

When two principals share a secret key they have access to a channel, and whenever a message arrives that can be decrypted by the key, we say that the channel says something. Each message includes some identification of the sender, as must be the case when several processes are multiplexed through the node key.

When a channel quotes a principal A saying s , for example by transferring the message $\text{read}(A, s)$, we have that $C | A \text{ says } s$. By (P2) this is equivalent with $C \text{ says } A \text{ says } s$, and the usual axioms apply.

References

- [Abadi94] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. Technical report 125. DEC SRC, Palo Alto, CA, June 1994. A preliminary version of this paper has appeared in the Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy.
- [BAN90] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, **8**(1):18–36, February 1990. Also available as DEC SRC Research Report 39, originally published February 28, 1989, and revised on February 22, 1990.
- [Burrows94] M. Burrows, M. Abadi, and R. Needham. The scope of a logic of authentication. Technical report 39 (Appendix). DEC SRC, Palo Alto, CA, 13 May 1994.
- [Gong90] L. Gong. *Cryptographic Protocols for Distributed Systems*. PhD thesis. University of Cambridge, UK, April 1990.
- [Lampson92] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, **10**(4):265–310, November 1992.
- [Liebl93] A. Liebl. Authentication in Distributed Systems: A Bibliography. *ACM Operating System Report*, **27**(4):31–41, October 1993.
- [Needham93] R. Needham. Cryptography and Secure Channels. In S. Mullender, editor, *Distributed Systems*, pages 531–541, 2nd edition. ACM Press, 1993.
- [Stallings95] W. Stallings. *Network and Internetwork Security*. Prentice-Hall, 1995.