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

Tage Stabell-Kulø Andrea Bottoni

2. November 1995
Revised 17. May 2003

Abstract

Authentication in Distributed Systems: Theory and Practice [9] 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 some tricky points.

This is the revised version.

Introduction

The purpose of this small technical report is solely to assist in the reading of [9]. Because this is a technical report explaining elements in a published paper, we do not claim that any novel aspects can be found herein.

On notation: "the paper" and "this paper" implies [9], and all references, such as "as described in Section 7" or "as shown in Figure 1", refer to the relevant section in the paper. Figures and tables in the text you are reading just now are clearly referenced as belonging to *this technical report*.

In "theory" it is simple to implement authorization (and hence authentication) correctly. PKI is far too often named as a panacea. The monitor is simply to compare the credentials supplied with the request with the requirements specified by the ACL. If there is a match, the request is granted; if there isn't the request is logged and rejected (in that order).

The following observation should demonstrate why such a view is far too simplistic: Traditionally, in order to withdraw money from your own bank account, you would have had to sign a receipt. The bank kept the original, while you got a copy for your own records. When your withdraw from an ATM some other "signatures" are in play; this paper provides the necessary vocabulary to discuss such settings.

This technical report is meant to be read as a companion to the paper; we believe that reading this technical report in isolation will not be very fruitful.

A good grasp on the inherit complexity of authentication is required in order to fully enjoy the issues considered in this paper. The best starting point is probably [10] or [2].

Theory

In this section we will discuss the axioms, their interpretation, and immediate consequences.

When a theory is used to obtain a better understanding of a system, it is crucial to determine whether the axioms match the system we try to model. The classical example is whether Euclidean geometry is a description of the "real" world or not. Before using geometry, we must examine the axioms and determine whether they reflect the system at hand. In particular, you must decide whether the fifth postulate in the first book (on parallel lines) holds for the system at hand: If you are calculating how much graniglia you need to tile the library in your Italian villa it probably does, but if you are calculating the trajectory of Mercury it does not. There is nothing within a theory that can guide you in making this decision; great care should be exercised before modeling a system with any theory.

Statements

Axiom S3 reads:

$$\vdash \big(A\ \textbf{says}\ s \wedge A\ \textbf{says}\ (s \supset s')\big) \supset A\ \textbf{says}\ s' \quad (S3)$$

This is modus ponens for **says**; when things are "said" we can expect them to have consequences. Or, in other words, we expect that (honest) principals understand their beliefs, and, just as important, believe the consequences of their beliefs. As an example, assume that this message is received (where A is shorthand for Alice and B for Bob):

$$\{B \Rightarrow A\}_{K_A^{-1}} \qquad (1)$$

We could now reason as follows: Since we know that K_A is Alice's key, we conclude that the statement is (was) made by Alice (or someone Alice trusts). She says that Bob speaks for her, and by (P10), this implies that we can proceed as if it is true that Bob speaks for Alice.

However, before we proceed there are two issues we should consider:

1. Is Alice honest?

   If she is not there will be problems, but the source of the problems will be obvious.

2. Does she know what she is doing?

   Message (1) can be viewed as a contract. In many countries contracts need to be signed in front of a Notarius Publicus. One of the reasons is that there is a need to determine not only who signed the contract (authentication), but that the signer understood what the contract implied. Or, in other words, in those countries it is not assumed that whenever something is said, we can proceed as if the consequences are true.

   Axiom (S3) makes it explicit that this theory should only be applied to systems where the participants understand the consequences of their actions.

BAN is another well-known theory that deals with settings similar to the one considered in this paper [3]. In BAN, honesty is implicitly assumed in that anyone who is believed to have jurisdiction over an issue (for example to generate a "good" encryption key), will do so. BAN has been criticized for this and a different semantics (than the possible-world) makes away with this; see [1] for a discussion. Honesty is not assumed in the paper we will discuss, but (S3) makes it clear that all participants are assumed to be rational. Assume that message (1) was received by Server, and it made the Server grant Bob access to Alice's resources. The crux of the matter is how Server will respond to the following accusation from Alice:

> "You received a random message from me, and you concluded that Bob is to be granted access to my resources. How could you draw this conclusion from this message? Please note that the message did *not* say "Please give Bob access to my resources"."

Can the server detect a difference between lying and lack of rationality?

Principals

It is taken as axioms in the theory that:

(P4): $\vdash \wedge$ is associative, commutative, and idempotent.

This means that it is taken for granted that the following holds (in the same order):

- $(A \wedge B) \wedge C = A \wedge (B \wedge C)$
- $A \wedge B = B \wedge A$
- $A \wedge A = A$

To us, all these seem reasonable.

(P6): $\vdash |$ distributes over $\wedge$ in both arguments.

This means that it is taken for granted that the following holds:

- $A|(B \wedge C) = (A|B) \wedge (A|C)$
- $(A \wedge B)|C = (A|C) \wedge (B|C)$

To us, both seem reasonable.

(P7): $\vdash (A \Rightarrow B) \equiv (A = A \wedge B)$

Two paragraphs later the following three conclusions are drawn (numbered by us):

> The operators $\wedge$ and $\Rightarrow$ satisfy the usual laws of propositional calculus. (1:) In particular, $\wedge$ is *monotonic* with respect to $\Rightarrow$.

This means that if $A \Rrightarrow B$ then $A \wedge C \Rrightarrow B \wedge C$. (2:) It is also easy to show that $|$ is monotonic in both arguments and (3:) that $\Rrightarrow$ is transitive.

Below we will first show that all these are formally correct, then give justifications supporting them.

Statement 1

When an operator is *monotonic* with respect to another (point in case: $\wedge$ is *monotonic* with respect to $\Rrightarrow$) the first can be applied to both sides of the second. Or, in other words, the following should hold:

$$(A \Rrightarrow B) \supset \big((A \wedge C) \Rrightarrow (B \wedge C)\big) \tag{2}$$

Formally is does. Assume that

$$A \Rrightarrow B$$

then (by (P7)

$$A \wedge C = (A \wedge B) \wedge C.$$

Because $\wedge$ is idempotent (by (P4) we get

$$A \wedge C = (A \wedge B) \wedge C \wedge C$$

and because $\wedge$ is associative (also by (P4)) we get

$$A \wedge C = (A \wedge C) \wedge (B \wedge C).$$

If we apply (P7) we obtain

$$A \wedge C \Rrightarrow B \wedge C.$$

Hence (2) holds.

The justification is that when $A \Rrightarrow B$ then the principal A "controls" B. Hence, when A utters a statement together with another principal, B will support it (together with the same principal, if need be).

Statement 2

That $|$ is monotonic in both arguments means that the following should hold:

$$(A \Rrightarrow B) \supset \big((A|C) \Rrightarrow (B|C)\big) \tag{3}$$

and

$$(A \Rrightarrow B) \supset \big((C|A) \Rrightarrow (C|B)\big) \tag{4}$$

Formally this holds; we will prove only (3) because the other is symmetric. Assume that

$$A \Rrightarrow B$$

then

$$A|C = (A \wedge B)|C.$$

By (P6) we have that

$$A|C = A|C \wedge B|C,$$

hence (by (P7))

$$A|C \Rrightarrow B|C.$$

The justification for (3) is that when B is "controlled" by A then when A quotes C, then B would also do so. The justification for (4) is similar. Assume a channel C. A message arrives on this channel, and the sender claims that the content is supported by A. Whenever this happens to be true, and if at the same time $A \Rrightarrow B$, the sender can claim that the message is supported also by B. For this reason, (4) is justified.

Please take care to understand that whether C is right when he quotes A or not, and hence whether this applies also to B, depends on the will and ability of C to ensure that he is acting correctly; it is not an aspect of this theory to ensure that no one is lying. Furthermore, knowing who C is does not ensure he is honest.

Statement 3

The question is if the following holds:

$$(A \Rrightarrow B) \wedge (B \Rrightarrow C) \supset A \Rrightarrow C.$$

Formally it does, because by using the axiom (P7) together with A, B and C we obtain:

$$A \Rrightarrow B \equiv A = A \wedge B \tag{5}$$

and

$$B \Rrightarrow C \equiv B = B \wedge C \tag{6}$$

by substituting (6) in (5) we obtain

$$A = A \wedge B \wedge C$$

which by (5) gives us

$$A = A \wedge C$$

which, according to the axiom, is equivalent to

$$A \Rightarrow C.$$

The justification is that if A speaks for B (B will, if need be, repeat everything A says), and B speaks for C in the same manner, then whatever A says will be supported by C.

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 content in advance. Or, in other words, it is not possible to use encryption without some A PRIORI agreement on syntax. “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 text[1] in the message. Furthermore, 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. Notice that a key by its nature is random, and it is (well: should be) impossible to distinguish between random data, encrypted data and a new key without resorting to syntax. This might not be necessary if the protocol is guaranteed to detect this fact later [6].

Let us turn now to how the functionality of public-key encryption can be implemented by the application of (only) shared-keys and a relay. The setting is one where B exchanges messages with R. As explained in Section 4.3, and shown in Table II, after the exchange B can believe that “K_a **says** s” even though B does not possess K_a.

First a few words on notation. The key K_a^b is key K_a together with an identifier (probably a number) that will be used to identify the key. The identification (in this case ^{b}) is local to the receiver, and the implementation of systems should be such that it will be detected without delay if an incorrect identificator is used.

[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 to “known text” attacks.

The starting point is that B receives from R the message:

$$K_b^b|K_a^r \textbf{ says } s \tag{7}$$

and the question is, how can B conclude that A **says** s?

As usual, we assume for simplicity that the certification authority is trusted by all participants:

$$K_{ca} \Rightarrow \texttt{anybody} \tag{8}$$

and

$$K_{ca} \textbf{ says } K_b \Rightarrow B.$$

B trusts the certification authority to having provided him and R with a key to share:

$$K_{ca} \textbf{ says } K_b \Rightarrow R$$

which implies, using (P11):

$$K_b \Rightarrow R \tag{9}$$

and

$$K_b \Rightarrow B$$

Using (9) inside (7) we obtain[2]:

$$R|K_a^r \textbf{ says } s \tag{10}$$

For the relay to work, R must be trusted by all principals; that is:

$$R \Rightarrow \texttt{anybody} \tag{11}$$

How this is achieved is not of interest to us here. If the system is designed in such a way that (11) is unacceptable, then delegation could be used instead.

Using (11) to simplify (10) we obtain

$$K_a^r \textbf{ says } s \tag{12}$$

Notice that although (12) is very similar to

$$K_a \textbf{ says } s \tag{13}$$

they are very different. Message (7), which contains a claim about (12), is received as a message encrypted with K_b and an identifier that tells B the message is intended for him (and hence sent by R

[2]See footnote 12 for the explanation.

```
(shared-key-encr (key-id 42) <ciphertext>)
```

Figure 1: Message sent from R to B, before decryption

```
(says (key-id 43) (statement <s>))
```

Figure 2: `<ciphertext>`, after decryption

and not B himself), while (13) would be a message encrypted with K_a itself. Message (7) could be encoded as shown in the Figures 1 and 2 (both in this technical report). Notice that 42 and 43 are not encryption keys but rather indices into a table where the keys can be found.

while message (12) could be encoded as in Figure 3 (in this technical report). Recall that B does not possess K_a; this is the rationale for the entire exercise.

Splicing the channels

Let us consider the following scenario: A is a server and B is a client that uses one of A's services. B wants to authenticate the messages coming from A; he knows that $K_a^r \Rightarrow A$. For every message that B receives from A (K_a^r **says** s) he needs to talk with the relay for asking and receiving the translation. This introduces performance and reliability issues. What R could do, instead, is to create a temporary shared key channel between A and B, by splicing the channels A-R and B-R. To that end, R generates a key K and two key identifiers K^a and K^b, to allow A and B to get hold of the key.

$$K^a = (K_a^a, \texttt{Encrypt}(K_a, K)$$

$$K^b = (K_b^b, \texttt{Encrypt}(K_b, K)).$$

Now, R can easily tell B about the key K, by saying:

$$K_b^b \textbf{ says } K^b \Rightarrow K_a^r. \qquad (14)$$

When B receives this message from R, he knows that he has a shared key channel K with the principal who holds K_a^r (which is A, since, from what we have assumed, B knows that $K_a^r \Rightarrow A$). Now the point is: how can R tell A about the key to use? In our scenario, in fact, only A's authentication is required: A answers the requests he receives regardless of the principal who is behind them. A possible solution is to delegate to B the task of telling A about the channel he has with B, i.e. about the key to use when answering to A's requests. This is easily achieved if B's request is:

$$K^{ab} \textbf{ says } s$$

where s is B's request and $K^{ab} = (K^a, K^b)$. A, by accessing K^a, gets hold of the key K and answers to B as follows:

$$K^b \textbf{ says } s$$

where s is now a reply message. R can easily tell B about K^a by including it in message (14), which becomes:

$$K_b^b \textbf{ says } K^{ab} \Rightarrow K_a^r.$$

Notice that, with this approach, neither A nor R need to keep any state.

Authenticating the channel

So far, the assumption in our scenario was that B knew $K_a^r \Rightarrow A$. How can B get to know this? The idea is to have a certification authority sign an equivalent of a public key certificate, i.e.

$$K_{ca}^x \textbf{ says } K_a^y \Rightarrow A$$

The crucial is: what should X and Y be?

The statement made by the certification authority should be both general and accessible by every principal. In order for the statement by the CA to be readable by every principal, it has to be X = R (If we put, for instance, X = B, C is not able to read it). This way, every principal can ask the relay to translate the statement made by the certification authority. In our case, B will send two messages to R

$$K_{ca}^r \textbf{ says } K_a^y \Rightarrow A$$

and

$$K_b^{br}$$

```
(signed-message
      (says (public-key K_a)(statement <s>))
      (signature <signature>))
```

Figure 3: Signed message sent from R to B

while R will reply with

$$K_b^b | K_{ca}^r \textbf{ says } K_a^y \Rightarrow A$$

and B will infer (since he trusts both the CA and R),

$$K_a^y \Rightarrow A.$$

B wants to be able to infer A **says** s from K_a^r **says** s, therefore he needs Y = R. For this reason, the CA will issue

$$K_{ca}^r \textbf{ says } K_a^r \Rightarrow A. \tag{15}$$

If we compare the certificate by the CA in the article with (15), we see that the latter has a missing key identifier, namely K_a^a. The reason why K_a^a is needed comes from the semantic of the certificate. The CA is guaranteeing that K_a is A's key, the one that A shares with R. In order to refer to the key without explicitly quoting it, the CA uses R's key identifier. Now, assume that A wants to check that the key he is using when speaking to relay R is really the one he is supposed to use (the one that has been certified by the CA), or assume that he wants to retrieve the key from the certificate by the CA.[3] A should be allowed to do so, and the only way to make it possible is to add his key identifier to the certificate, which becomes:

$$K_{ca}^r \textbf{ says } K_a^{ar} \Rightarrow A$$

as in the article.

Splicing and authenticating

In the previous paragraphs we have seen how a relay R can splice two channels into a channel K and how R can translate a statement by the CA in order to authenticate a principal (A). In order to do so, the statements by R are, respectively:

$$K_b^b \textbf{ says } K^{ab} \Rightarrow K_a^r \tag{16}$$

$$K_b^b | K_{ca}^r \textbf{ says } K_a^{ar} \Rightarrow A. \tag{17}$$

R could do both things together, i.e. he could authenticate the spliced channels. If B receives both (16) and (17), since he trusts both R and the CA, he understands

$$K^{ab} \Rightarrow K_a^r$$

and

$$K_a^{ar} \Rightarrow A.$$

Then, by applying the transitivity property of $\Rightarrow$, he gets

$$K^{ab} \Rightarrow A$$

R could also derive the same conclusions by himself and say (16) and (17) in one mouthful as follows:

$$K_b^b | K_{ca}^r \textbf{ says } K^{ab} \Rightarrow A \tag{18}$$

Again, (18) is obtained by using (16) inside (17) and applying the transitivity property of $\Rightarrow$.

Two way authentication

If A has the same requirements for authentication as B has, then R needs to produce symmetric statements regarding B. Therefore, when splicing the channels, R will send to A:

$$K_a^a \textbf{ says } K^{ab} \Rightarrow K_b^r$$

When splicing and authenticating at the same time, R will fetch B's certificate by the CA:

$$K_{ca}^r \textbf{ says } K_b^{br} \Rightarrow B$$

and will tell A about the new channel he has with B as follows:

$$K_a^a | K_{ca}^r \textbf{ says } K^{ab} \Rightarrow B.$$

[3]This only works if the key identifier contains the key encrypted with the principal's master key.

Principals with names

A single Certification Authority

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

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 line. We want to construct a scheme where we have an agent that is on line supporting CA which is off line. In other words, 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:

$$K_{ca} \textbf{ says } (O|K_a \wedge K_a) \Rightarrow A \qquad (19)$$

(we will return in a moment to the "meaning" of this certificate). O then counter-signs this by issuing

$$O|K_a \textbf{ says } K_a \Rightarrow O|K_a \qquad (20)$$

From these two, $K_{ca} \Rightarrow A$, and (P12) we again get $K_a \Rightarrow A$ if we proceed as follows: we start with (19); in prose, it is:

> "K_{ca} assert that $K_a \Rightarrow A$, if O also says so".

We assume that $K_{ca} \Rightarrow A$, and with (19) and (P8) yields

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

(P10) is used to obtain

$$O|K_a \wedge K_a \Rightarrow A \qquad (21)$$

In prose, (20) 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, (20) with (P10) gives

$$K_a \Rightarrow O|K_a \qquad (22)$$

The meaning of (22) becomes less obscure if we use (P7) and substitute K_a for A and $O|K_a$ for B. Then we obtain

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

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

We now substitute (22) in (21) and get

$$K_a \wedge K_a \Rightarrow A$$

which, by (P4), gives us the desired

$$K_a \Rightarrow A.$$

Path Names and Multiple Authorities

It it 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 trust 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

$$\texttt{/dec} \textbf{ says } \texttt{/} \Rightarrow \texttt{/dec} \qquad (23)$$

so that we can believe that

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

when

$$\texttt{/} \textbf{ says } K_{\texttt{/dec}} \Rightarrow \texttt{/dec}$$

and

$$\texttt{burrows} \textbf{ says } \texttt{/dec} \Rightarrow \texttt{burrows}$$

so that we can believe that

$$K_{\texttt{burrows}} \Rightarrow \texttt{burrows}$$

when

$$\texttt{/dec} \textbf{ says } K_{\texttt{burrows}} \Rightarrow \texttt{burrows}$$

we end up with

$$/ \Rightarrow \texttt{burrows} \tag{24}$$

since $\Rightarrow$ is transitive. The problem is that DEC might not accept (23) even though DEC is part of the tree, and burrows might find (24) not to be true regardless of whether he works for DEC. Furthermore, assume Alice issues the following certificate:

$$\texttt{alice}\ \textbf{says}\ K_{\texttt{alice}} \Rightarrow \texttt{alice}$$

Incidently, Alice happens to be root of the tree, but how is burrows going about to establish this fact? It would have been nice to have

$$K_{\texttt{dec}}\ \textbf{says}\ K_{\texttt{alice}} \Rightarrow \texttt{root}$$

and, vice versa:

$$K_{\texttt{alice}}\ \textbf{says}\ \texttt{dec} \Rightarrow \texttt{root} \tag{25}$$

in order for burrows to use (P10) and deduce that

$$K_{\texttt{alice}} \Rightarrow \texttt{root}$$

The problem is that Alice might very well find it somewhat inconvenient to issue (25).

We need a mechanism enabling us to model a tree, where trust is localized but (nevertheless) transitive. Furthermore, in a tree, the relations are different upwards and downwards: this should be visible. This means that it should be possible to distinguish a 'downward pointing' certificate from an 'upward pointing'. The underlying problem is twofold: handoff is unconditional and "$\Rightarrow$" is transitive. And as we have seen, this is an issue both 'up' and 'down' in the tree. We will walk through an example before embarking on a discussion on these mechanisms in general, and alternatives.

We start with burrow's assumption about himself and his position in the tree:

$$C_{\texttt{burrows}} \Rightarrow \texttt{/dec/burrows}\ \textbf{except}\ \texttt{nil} \tag{26}$$

which by the monotonicity of quoting is equal to

$$C_{\texttt{burrows}}|\texttt{`..'} \Rightarrow (\texttt{/dec/burrows}\ \textbf{except}\ \texttt{nil})|\texttt{`..'}. \tag{27}$$

We also have that, by (N3), the right hand of this[4] implies that:

$$(\texttt{/dec/burrows}\ \textbf{except}\ \texttt{nil})|\texttt{`..'} \Rightarrow \texttt{/dec}\ \textbf{except}\ \texttt{burrows}$$

[4]Please notice that the very last period in (27) terminates the sentence and is not part of the equation itself.

which, together with (26) and (27) demonstrates that

$$C_{\texttt{burrows}}|\texttt{`..'} \Rightarrow \texttt{/dec}\ \textbf{except}\ \texttt{burrows}. \tag{28}$$

In this position, a certificate is then issued:

$$C_{\texttt{burrows}}|\texttt{`..'}\ \textbf{says} C_{\texttt{dec}} \Rightarrow \texttt{/dec}\ \textbf{except}\ \texttt{burrows} \tag{29}$$

which, by (28) and (P8), is equal to

$$\texttt{/dec}\ \textbf{except}\ \texttt{burrows}\ \textbf{says} C_{\texttt{dec}} \Rightarrow \texttt{/dec}\ \textbf{except}\ \texttt{burrows}$$

and by using (P10) we get the desired

$$C_{\texttt{dec}} \Rightarrow \texttt{/dec}\ \textbf{except}\ \texttt{burrows} \tag{30}$$

which is what burrows needs in order to know something about his "parent" node in the tree.

There are several issues worth discussing regarding these axioms and their application. The first is whether they are needed at all. Basically the concept of **except** captures that it is not unimportant who actually says what (not all parties are equal); the position in the tree matters. Another approach would be to introduce a "speaks for regarding" relation, where the "regarding" would be "statements regarding positions in the tree". Unfortunately the relation "speaks for regarding" has deep-rooted problems also under very reasonable semantics; please see the section named "Discussion" at the end of this technical report.

The aim of the entire endeavor is for burrows to obtain the credential

$$C_{\texttt{mit}}|\texttt{clark}\ \textbf{says} C_{\texttt{clark}} \Rightarrow \texttt{/mit/clark}\ \textbf{except}\ \texttt{`..'}. \tag{31}$$

It is reasonable to ask why mit simply do not issue this certificate, and the usual axioms would apply? If clark is to issue certificates for those below him in the tree, their credentials would be

$$C_{\texttt{mit}}|C_{\texttt{clark}}|\texttt{alice}\ \textbf{says}\ C_{\texttt{alice}} \Rightarrow \texttt{/mit/clark/alice}\ \textbf{except}\ \texttt{`..'}$$

and so on; this chain of quotes is not at all esthetically appealing. The combination of **except** and quote together maintains the fact that there is a difference between a node speakign about a subordinate, and one stating his own position in the tree.

Another point of importance is that burrow's view of mit's rôle (as external to mit) is quite different from clark's (as internal). Also this aspect is captured by the credentials that are in play. This becomes evident when we notice that while burrows infers

$$C_{\mathtt{mit}}|\mathtt{clark}\ \textbf{says}\ C_{\mathtt{clark}} \Rightarrow \mathtt{/mit/clark}\ \textbf{except}\ \text{`}\mathtt{..}\text{'}$$

to understand clark's position, clark uses

$$C_{\mathtt{clark}}|\text{'}\mathtt{..}\text{'}\ \textbf{says}\ C_{\mathtt{mit}} \Rightarrow \mathtt{/mit}\ \textbf{except}\ \mathtt{clark}$$

on exactly the same relation. If we issued certificates like (31), rather than infering credentials as we have shown, this difference would vanish.

This last point also highlights the underlying machinery: The path upwards in a tree (from your own position) is different from that of a peer descending down towards you, even if the nodes that are "visited" are identical.

Roles and Programs

Rôles are used to restrict authority. There are two reasons why rôles are useful (and hence desirable): first and foremost rôles are used to make it possible to restrict authority. Alice can delegate authority over the rôle "Alice **as** User" without having to relinquish authority over "Alice **as** Manager", or the all-powerful "Alice".

Secondly, one could believe that, based on by (R1) and (R2), rôles are only shorthand for quoting. This is not the case, since the operator **as** is strictly typed: the axioms only hold when the right-hand side is a rôle. The problem is that building proofs that contains arbitrary combinations of $\wedge$ and $|$ can take exponential time. As a higher-level operator, **as** can be combined in fewer ways, hence keeps the problem tractable.

Also notice the implementation issues stemming from the fact that rôles are part of a global name space. That is, assume we have the following two certificates:

$$A\ \textbf{says}\ B \Rightarrow (A\ \textbf{as}\ \mathtt{user})$$

and

$$P\ \textbf{says}\ B \Rightarrow (P\ \textbf{as}\ \mathtt{user})$$

How do we know that A and P have the same understanding of the rôle user? Even though this is related to an implementation, it is not a detail.

Loading Programs

If A doesn't trust the file system, he 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 authority is being handed over to another principal, e.g. A **says** $B \Rightarrow A$ there are two issues to consider:

- The handoff is unconditional. If a certificate that hands off authority needs to be verified, an online certification service must be added as discussed in Section 5.1.
- When A hands off his authority to B there is no difference between A and B since, by (P7), we have that $(B \Rightarrow A) \equiv (B = B \wedge A)$; however, see below for a discussion. 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. Now, B might not care that A suffers—after all it was A how handed off his authority, not B—but B might suffer as well. In particular, others who known that $B \Rightarrow A$ might expect B to know, and to behave accordingly. But how, in the general case, is B to know that it speaks for A? Delegation is less strict than handoff in that a delegation must be acknowledged.

Before we proceed, let us digress a moment to discuss the semantics of $B \Rightarrow A$. Under the semantics based on "possible worlds", A would allow B to speak for him, only when the set of worlds believed possible by B is a subset of those believed possible by A; if there are worlds believed to be possible by B but not by A, it is impossible for A to let B speak for him since B might enter a world which is impossible for A. An important consequence of this semantics is that A might consider worlds possible that B will never enter; A set of worlds might be larger than those of B. Thus, even though B speaks for A, A might say things not supported by B. Or, with other words: it is too simplistic to claim that A and B are *equal* when $B \Rightarrow A$.

Given the axiom

$$\vdash A \wedge B|A \Rightarrow B \textbf{ for } A \tag{D1}$$

and the two certificates

$$A \textbf{ says } B|A \Rightarrow B \textbf{ for } A \tag{32}$$

and

$$B|A \textbf{ says } B|A \Rightarrow B \textbf{ for } A \tag{33}$$

we obtain

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

and by (D1) we get

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

which by (P10) yields

$$B|A \Rightarrow B \textbf{ for } A.$$

The crucial issue becomes visible in the conclusion, since only when B explicitly quotes A does it speak for B **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.

Axioms for Delegation

As pointed out in the article in Footnote 24, a different set of axioms could have been used as the basis of delegation:

$$\vdash B \textbf{ for } A = B|A \wedge D|A \tag{D5}$$
$$\vdash A \Rightarrow D|A \tag{D6}$$

These two axioms are more powerful than (D1) and (D2). In fact, using only (D5) and (D6) is possible to derive (D1)–(D4) as theorems.

As an example, we will derive here (D1): $A \wedge B|A$ can be written, using (D6) and (P7) as

$$A \wedge D|A \wedge B|A$$

Since by (P4) $\wedge$ is idempotent and commutative, this is equal to

$$A \wedge B|A \wedge B|A \wedge D|A.$$

Now,

$$A \wedge B|A = A \wedge B|A \wedge B|A \wedge D|A$$

can be written, using (P7) as

$$A \wedge B|A \Rightarrow B|A \wedge D|A$$

and by using axiom (D5) we get to (D1):

$$A \wedge B|A \Rightarrow B \textbf{ for } A.$$

Let's turn now to the meaning of (D5) and (D6). A wants to delegate to B in a way that ensures that B's identity is visible in requests made by B on A's behalf. Therefore handoff (A **says** $B \Rightarrow A$) is not appropriate. A new compound principal needs to be introduced: B **for** A. A could use a delegation service by the trusted principal D, to manage giving and revoking the delegation. A could tell D about his delegation to B (by some means) so that, whenever B|A says something, D|A, triggered by B, says the same. B|A **says** s and D|A **says** s, together give $(B|A \wedge D|A)$ **says** s, which becomes, using (D5) (B **for** A) **says** s.

In this setting D is a critical server: it must be both highly available and secure; therefore it may not be convenient to implement it. A more realistic approach could be to allow the principals to be their own delegation service. In other words, we are introducing axiom (D6) in our model of reality.

What has changed now is the fact that, in order for B **for** A to say something, we need both B|A and A to say so. This is not what delegation is for: A would better not be involved in confirming every statement by B. He will rather delegate to B the right to speak for B **for** A. In order to do so, he will produce the certificate:

$$A \textbf{ says } \mathsf{B}|\mathsf{A} \Rightarrow \mathsf{D}|\mathsf{A}. \tag{34}$$

As long as this certificate is valid, B **for** A = B|A, therefore A has to be careful in estimating its lifetime.[5]

There is an important drawback in this approach: B does not acknowledge the delegation. Therefore, instead of certificate (34), A will propose the delegation to B as follows:

$$A \textbf{ says } \mathsf{B}|\mathsf{A} \Rightarrow \mathsf{B} \textbf{ for } A \tag{35}$$

and B will acknowledge it:

$$\mathsf{B}|\mathsf{A} \textbf{ says } \mathsf{B}|\mathsf{A} \Rightarrow \mathsf{B} \textbf{ for } A. \tag{36}$$

Summing up, we can say that (D5) and (D6) are the basic axioms for delegation. (D5) models the concept of delegation, while (D6) models the shift of the delegation service towards the principals. Using only these two axioms, we can have (D1)–(D4) as theorems. On the other hand, (D5) and (D6) lead to a delegation certificate by A which is too powerful (it does not need B's awareness) and involves a fake principal D. Instead, (D1) is what we need to use for having delegation when certificates as (35) and (36) are used, while (D2)–(D4) are what we need for the manipulation of expressions with operator **for**. This is why for practical purposes we will consider (D1)–(D4) as our axioms for delegation.

Login

When the user logs in, the user's key K_u must be used to sign certificates that are used to demonstrate that the user is logged in; more on this below. In the case a user actually has a public-secret key-pair login proceeds as described in the paper; both keys, or at least the private one, must be stored at a safe location such as on a smartcard. Even though a malicious workstation can still wreck havoc on the objects the user controls, the login will eventually time out and no further harm can be done; the mechanisms for this are discussed below.

When authentication is done with passwords the user is in a less favorable position. If we assume that the password (or phrase) is used as an encryption key (possibly after hashing it to a proper length), this key then speaks for the user. This key is then used to sign the certificate (37) shown (and discussed) below. The issue here is that when the user has given his password to the workstation, $W \Rightarrow \mathsf{K}_u$. The workstation will proceed as usual (as discussed below), but there si a significant difference: Since K_u is a shared key, the workstation is not able to provide evidence that the user *is* logged in (as opposed to *has been*). This is equivalent to saying that the login certificate (37) does not time out; there is no time limit on the malicious behavior of a workstation.

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

$$\mathsf{K}_u \textbf{ says } (\mathsf{K}_w \wedge \mathsf{K}_l)|\mathsf{K}_u \Rightarrow \mathsf{K}_w \textbf{ for } \mathsf{K}_u \tag{37}$$

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). Notice that K_l is a newly generated, temporary, public key.

Because | distributes over $\wedge$ we can write (37) as

$$\mathsf{K}_u \textbf{ says } (\mathsf{K}_w|\mathsf{K}_u \wedge \mathsf{K}_l|\mathsf{K}_u) \Rightarrow \mathsf{K}_w \textbf{ for } \mathsf{K}_u$$

and we can clearly see that either K_w and K_l *together* must issue all statements (and explicitly quote the user), or K_w must obtain the power to speak for K_l.

The workstation now accepts (his part of) the delegation, and acknowledges that it must cooperate with the session key:

$$\mathsf{K}_w|\mathsf{K}_u \textbf{ says } (\mathsf{K}_w \wedge \mathsf{K}_l)|\mathsf{K}_u \Rightarrow \mathsf{K}_w \textbf{ for } \mathsf{K}_u \tag{38}$$

[5]One of the reason for having a dedicated revocation service was to manage revocation of delegation.

which in prose is:

> I (i.e. K_w) acknowledge that I am aware that the user K_u has said that whenever I and K_l together quote him, together we speak for the combined principal K_w **for** K_u.

In order for the delegation to be valid, also K_l must acknowledge (37) in the same manner as K_w did in (38). However, the rôle of K_l is "only" to act as a device to ensure that the delegation is limited in time, and there is no need for K_l to participate in the exchange of messages. Hence, K_l will hand off all its power to K_w, and in order to do so, issues:

$$K_l \textbf{ says } K_w \Rightarrow K_l. \tag{39}$$

Armed with (39), K_w is as powerful as K_l is; recall that according to (P7) we have that $A \Rightarrow B \equiv A = A \wedge B$. Since $K_w \Rightarrow K_l$ it should be apparent that $K_w|K_u \Rightarrow K_l|K_u$. For this reason, there is no need for K_l to issue a certificate similar to (38). Following this line of argument, it becomes clear why (38) and (39) together accepts the delegation offered in (37).

Note that (39) is an optimization and as such should 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. How this is done is irrelevant as long as the identification is unique.

When a channel quotes a principal A saying s, for example by transferring the message `read(A, s)`, we have that $C|A$ **says** s. By (P2) this is equivalent to C **says** A **says** s, and the usual axioms apply. Recall that C might lie or be mistaken.

As an example of how this is used, Figure 7 shows an expanded version of the example in Figure 1. The issue at hand is to determine which credentials must be supplied with a request to read a file in a non-trivial setting. It is assumed that the request is simply "read `foo`", that the request is sent on the (encryption) channel C, and the request is somehow (by C) marked as originating from the process running the `Accounting` application; we name this process `pr`.

The ACL attached to the file contains "`src-node` **as** `Accounting` **for** `bwl` may read". We understand this to be: "A machine that can provide evidence it is configured such that it belongs to the group of machines named `src-node`, and where the `Accounting` application is running on behalf of the user `bwl`, may read when the request is made by the user."

Three important credentials were generated as side effects of booting the machine and `bwl`'s login. They are shown in Table 1 (in this technical report).

In addition to the credentials that were generated as part of booting and login, there is a set of "standard" credentials that are (assumed to be) available to the server, either by caching or by accessing a database of credentials. These are shown in Table 2 (in this technical report).

By combining the credentials in Table 1 (in this technical report) and Table 2 (in this technical report) the server is able to derive a set of relationships between the principals that are involved.

All the credentials in Tables 1 (in this technical report) and 2 (in this technical report) are not sufficient to accommodate the request. In particular, although the server believes `bwl` is logged in on `ws`, there is nothing tying the principals together. The ACL requires that the machine, the operating system, the application and the user all be asserted to be operating in concert.

One possible solution would be that K_n creates a channel C, and issues a certificate asserting that the channel speaks for the combined principal "`src-node` **as** `Accounting` **for** `bwl`". The question is: is it in the power of K_n to do so? It is, if we can prove that:

$$K_n|K_{\texttt{bwl}} \Rightarrow (\texttt{src-node} \textbf{ as } \texttt{Accounting}) \textbf{ for } \texttt{bwl}$$

From booting `ws` it follows that

$$k_n \Rightarrow K_{\texttt{ws}} \textbf{ as } \texttt{Taos}$$

and because **as** is monotonic,

$$K_n \textbf{ as } \texttt{Accounting} \Rightarrow (K_{\texttt{ws}} \textbf{ as } \texttt{Taos}) \textbf{ as } \texttt{Accounting}.$$

Credential	Comment	
K_{ws} **says** $K_n \Rightarrow K_{ws}$ **as** `Taos`	From booting `ws`	(C1)
K_{bwl} **says** $(K_n \wedge K_l)\vert K_{bwl} \Rightarrow K_n$ **for** K_{bwl}	From `bwl`'s login	(C2)
K_l **says** $K_n \Rightarrow K_l$	Also from login	(C3)

Table 1: New Certificates

Credential	Comment	
$K_{ws} \Rightarrow$ `ws`	HW	(C4)
`ws` **as** `Taos` $\Rightarrow$ `src-node`	HW + OS	(C5)
$K_{bwl} \Rightarrow$ `bwl`		(C6)

Table 2: Established certificates

$K_n \Rightarrow K_n$ **as** `Accounting`, since `Accounting` is a rôle. Using the transitivity property of $\Rightarrow$, we get

$$K_n \Rightarrow (K_{\mathtt{ws}} \textbf{ as } \mathtt{Taos}) \textbf{ as } \mathtt{Accounting}.$$

Since **for** is monotonic we get

$$K_n \textbf{ for } K_{\mathtt{bwl}} \Rightarrow \big((K_{\mathtt{ws}} \textbf{ as } \mathtt{Taos}) \textbf{ as } \mathtt{Accounting}\big) \textbf{ for } K_{\mathtt{bwl}}.$$

Now, since $K_{\mathtt{bwl}}$ has delegated authority to K_n, we have that $K_n|K_{\mathtt{bwl}} \Rightarrow K_n$ **for** $K_{\mathtt{bwl}}$. Using the transitivity property of $\Rightarrow$, we get

$$K_n|K_{\mathtt{bwl}} \Rightarrow \big((K_{\mathtt{ws}} \textbf{ as} \mathtt{Taos}) \textbf{ as } \mathtt{Accounting}) \textbf{ for } K_{\mathtt{bwl}}. \quad (40)$$

What we need now is to show that

$$\big((K_{\mathtt{ws}} \textbf{ as} \mathtt{Taos}) \textbf{ as } \mathtt{Accounting}\big) \textbf{ for } K_{\mathtt{bwl}} \Rightarrow (\mathtt{src\text{-}node} \textbf{ as } \mathtt{Accounting}\big) \textbf{ for } \mathtt{bwl}.$$

Using $K_{\mathtt{ws}} \Rightarrow$ `ws` and **as**'s monotonicity, we get

$$K_{\mathtt{ws}} \textbf{ as } \mathtt{Taos} \Rightarrow \mathtt{ws} \textbf{ as } \mathtt{Taos}.$$

But `ws` **as** `Taos` $\Rightarrow$ `src-node`, therefore, since $\Rightarrow$ is transitive,

$$K_{\mathtt{ws}} \textbf{ as } \mathtt{Taos} \Rightarrow \mathtt{src\text{-}node}$$

Using, again, **as**' monotonicity, we obtain

$$(K_{\mathtt{ws}} \textbf{ as } \mathtt{Taos}) \textbf{ as } \mathtt{Accounting} \Rightarrow \mathtt{src\text{-}node} \textbf{ as } \mathtt{Accounting}$$

Using the monotonicity of **for**:

$$\big((K_{\mathtt{ws}} \textbf{ as } \mathtt{Taos}) \textbf{ as } \mathtt{Accounting}\big) \textbf{ for } K_{\mathtt{bwl}} \Rightarrow (\mathtt{src\text{-}node} \textbf{ as } \mathtt{Accounting}) \textbf{ for } K_{\mathtt{bwl}}$$

Because $K_{\mathtt{bwl}} \Rightarrow$ `bwl` and since **for** is monotonic, we have

$$(\mathtt{src\text{-}node} \textbf{ as } \mathtt{Accounting}) \textbf{ for } K_{\mathtt{bwl}} \Rightarrow (\mathtt{src\text{-}node} \textbf{ as } \mathtt{Accounting}) \textbf{ for } \mathtt{bwl}$$

Putting the last two equations together, by the transitivity of $\Rightarrow$ we get

$$\big((K_{\mathtt{ws}} \textbf{ as } \mathtt{Taos}) \textbf{ as } \mathtt{Accounting}) \textbf{ for } K_{\mathtt{kwl}} \Rightarrow (\mathtt{src\text{-}node} \textbf{ as } \mathtt{Accounting}\big) \textbf{ for } \mathtt{bwl}. \quad (41)$$

By using (40) and, again, the transitivity of $\Rightarrow$, we finally get to:

$$K_n|K_{\mathtt{bwl}} \Rightarrow (\mathtt{src\text{-}node} \textbf{ as } \mathtt{Accounting}) \textbf{ for } \mathtt{bwl}.$$

We have now proven that the server should accept the credentials of the channel C, and the process running the accounting application can now be allowed to use this channel.

There are (probably) many processes running on `ws`, and many of them might be "speaking" to the server. For simplicity, communication from all the processes will be multiplexed on C. Thus, it is necessary that each of these multiplexed channels is

```
(says <kbwl>
    (speaks-for
       (quote <C> <pr>)
       (for
          (as (as <kws> <Taos>) Accounting)
          <Kbwl>)))
```

Figure 4: Message sent on C

clearly identified, and is assigned the correct credentials. The process running the `Accounting` application is named `pr`, hence the channel we are interested in is C|`pr`. This channel needs to speak for `src-node` **as** `Accounting` **for** `bwl` for its requests to be honored. According to the paper, the server is able to deduce this credential from a message on the shared channel C, which has the following credentials:

$$K_n|K_{\mathtt{bwl}}\ \textbf{says}\ C|\mathrm{pr} \Rightarrow \big((K_{\mathtt{ws}}\ \textbf{as}\ \mathtt{Taos}\ )\ \textbf{as}\ \mathtt{Accounting}\big)\ \textbf{for}\ K_{\mathtt{bwl}} \tag{42}$$

The question is what this message "says" and how it is encoded.

The content of the message is: when a request arrives on the channel C quoting process `pr`, it has the credentials of

$$((K_{ws}\ \textbf{as}\ \mathtt{Taos})\ \textbf{as}\ \mathtt{Accounting})\ \textbf{for}\ K_{bwl}$$

Because of statement (41) and the transitivity of ⇒, it also means that C|`pr` has the credentials of

$$(\mathtt{src\text{-}node}\ \textbf{as}\ \mathtt{Accounting})\ \textbf{for}\ \mathtt{bwl}.$$

The server will believe it, because of (40) and the handoff rule.

Turning now to how the message is encoded: the message will appear on channel C, since $C \Rightarrow K_n$. To highlight this, we rewrite (42) as follows:

$$C\ \textbf{says}\ K_{\mathtt{bwl}}\ \textbf{says}\ C|\mathrm{pr} \Rightarrow \big((K_{\mathtt{ws}}\ \textbf{as}\ \mathtt{Taos}\ )\ \textbf{as}\ \mathtt{Accounting}\big)\ \textbf{for}\ K_{\mathtt{bwl}}. \tag{43}$$

The actual message on channel C would be the encoding of everything that appears after "C **says**" in statement (43). A possible encoding is shown in Figure 4 (in this technical report).

Discussion

The theory in this article lacks a semantics; there is no discussion about what the different constructions "mean". An effort to remedy the situation is presented in [7]. There it is made clear that assigning a semantics reveals subtle but important problems. In particular, assume that the "speaks for" operator is limited to a "speaks for regarding"-operator where $A \overset{T}{\Rightarrow} B$ means that A speaks for B on statements in the set T. The problem is that with the semantics that seems natural (using the same possible-world setting as in this paper) we get the surprising relation:

$$(A \overset{S}{\Rightarrow} B) \wedge (A \overset{T}{\Rightarrow} B) \not\supset (A \overset{S\cup T}{\Longrightarrow} B)$$

This indicates that a limited version of "speaks for" might require a more detailed investigation.

Another approach is taken by SPKI [4, 5]. There, certificates can be augmented with a "no delegation bit"; this captures the fact that the meaning of "speaks for" is dependent on the object. On the surface, when this bit is set it simply says that the rights granted in the certificate can not be delegated (further). However, in effect a new operator "speaks for but not delegatable" has been introduced: we write this as $\rightarrow\!\!|$. Now we face the problam that transitivity of "speaks for" ceases to hold; see the discussion regarding "Statement 3" on page 3 in this technical report.

The relationship between ⇒ and $\rightarrow\!|$ is the following:

$$\begin{aligned}(A \Rightarrow B) \wedge (B \rightarrow\!\!| \ C) &\supset A \rightarrow\!\!| \ C\\ (A \rightarrow\!\!| \ B) \wedge (B \Rightarrow C) &\not\supset A \Rightarrow C\\ (A \rightarrow\!\!| \ B) \wedge (B \Rightarrow C) &\not\supset A \rightarrow\!\!| \ C\end{aligned}$$

and this would complicate all the proofs we have carried out.

Based on these two observations we can only emphasize the same argumentas before: Before any theory can be used to reason about a (real) system, the validity of the axioms must be examined carefully; see the discussion on axioms in the section on Statements, on page 2 in this technical report.

References

[1] Martín Abadi and Mark Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[2] Ross J. Anderson. *Security Engineering*. John Wiley & Sons, Inc., 2001. ISBN 0-471-38922-6.

[3] Michael Burrows, Martín Abadi, and Roger 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.

[4] Carl Ellison. SPKI requirements. RFC 2692, The Internet Society, September 1999.

[5] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. Rfc, The Internet Society, September 1999.

[6] Li Gong and Paul Syverson. Fail-Stop Protocols: An Approach to Designing Secure Protocols. In *Proceedings of the 5th IFIP Working Conference on Dependable Computing for Critical Applications*, Urbana-Champaign, Illinois, September 1995.

[7] Jon Howell and David Kotz. A formal semantics for SPKI. Technical Report 2000-363, Department of Computer Science, Dartmouth College, March 2000. Extended version of [8]. Available from `http://www.cs.dartmouth.edu/reports/abstracts/TR2000-363/`.

[8] Jon Howell and David Kotz. A formal semantics for SPKI. In *Proceedings of the Sixth European Symposium on Research in Computer Security (ESORICS 2000)*, volume 1895 of *Lecture Notes in Computer Science*, pages 140–158. Springer-Verlag, October 2000.

[9] Butler Lampson, Martín Abadi, M. Burrows, and E. Wobber. Authentication in distributed

systems: Thory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.

[10] Richard E. Smith. *Authentication*. Addison-Wesley, 2002.