Authorization and Access Control in a Distributed File Repository

Ronny Arild and Tage Stabell-Kulø
Department of Computer Science
N-9037 University of Tromsø, Norway
{ronnya,tage}@pasta.cs.uit.no

CS-TR 99-34
January 18, 1999

Abstract

A distributed file repository is described. It supports interaction between different machines used by a single user, as well as between users that share data. Files can be replicated and consistency will be maintained, or files can be *shipped* (copied) to a remote site.

As with more traditional systems, the servers are trusted not to leak information. However, the rôle servers play is not as much the hub in the system. In particular, users are in charge of delegating acccess to files. For flexibility, delegations might take place outside of the realm of the system proper; by any means available to the users.

Users can delegate access rights to local and remote users, including remote users in other domains. ACLs are used to maintain local access control; capabilities are used to access remote files. These capabilities are valid within epochs, but are immediately revoked when being used, thus being valid at-most-once.

In essence, we have realized a flexible infrastructure where users can implement their own security policy.

1 Introduction

Traditional systems, such as Kerberos [Kohl and Neuman, 1993], are built around a system model that in our view no longer holds. It is simply not true that "clients" are solely placed in laboratories, and that users are dependent on servers in order to work. Powerful workstations and laptop computers have forced a shift in focus. Servers are still useful, but their rôle have changed; this fact should be reflected in the way supporting infrastructure

is designed. In particular, policy decisions are to a much greater extent made by users. Also, communication channels outside the system proper must be acknowledged in the sense that users can exploit such channels when possible (or desired). For example, in Kerberos, there is no way a user can grant others access to any of his files without interacting with Kerberos (except by giving away his password). Or, put in another way, the system model does not include enough trust in users to reign over his own files.

When, on the other hand, users are placed in the center of the computing model, new challenges must be solved. In this paper, it will be shown that regarding users as "first class citizens" have profound security implications. In particular, users might delegate authority (ie., grant access) to his files by means of his personal computers, without interacting with the servers that actually stores the files. Enforcing security policy now becomes an issue of implementing the users' policy. In this respect, our system is concerned with the proper handling of delegation in concert with replication.

This paper describes our system, with focus on how secure delegation of access is designed and implemented. Section 2 gives an overview of the system, the design rationale and some examples of its use. We focus on security by describing the setting where delegation takes place, and the obstacles our solution overcomes. Then, in Section 3 we cover the machinery put in place to enable delegations across administrative domains. Section 4 analyzes the protocols described in the previous section. In Section 5 we discuss our work. Finally, a conclusion is given in Section 6.

2 System overview

Users have a wealth of extra-system information. Such information range from well defined messages such as "Bob is away for the weekend" to more subtle and uncertain knowledge about the habits of colleagues. It is hard, if not impossible, to compile such knowledge into a form that can be utilized by systems. Humans, on the other hand, are able to consider the implications, and act accordingly. For example, if Bob is away for the weekend, he will most certainly not edit any files. In other words, if Alice shares some files with Bob, she can most probably alter them without taking any steps to protect herself against any consistency problems. Furthermore, *users* normally have access to communication channels outside the system proper, channels that can overcome network partitions (telephone is the prime example).

We have designed and implemented a system that is flexible enough to accommodate extra-system communication, while at the same time maintaining consistency on the data entrusted to it. Allowing users flexibility

has been our foremost goal. We also acknowledge that a contemporary distributed system is not homogenous. Hence, one can not assume that all users will run one particular operating system. Thus, the interface to our system is not an API but rather a protocol. The protocol, called FRTP (File Repository Transport Protocol, see below for details) is designed with flexibility in mind.

We now briefly describe our system, our research vehicle, as it were. We then outline the main theme of this paper, the means to delegate access to files to users in other administrative domains.

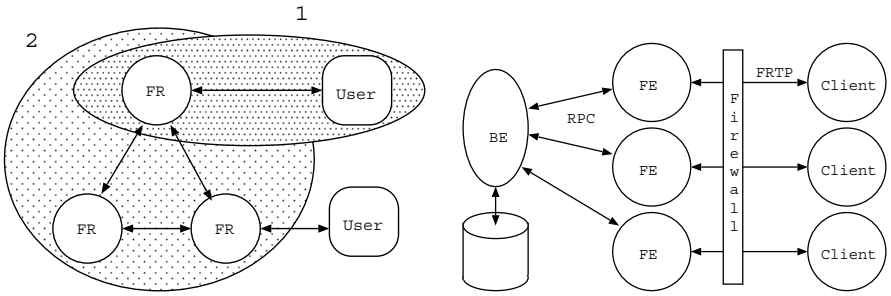
2.1 File Repository

We have designed and implemented a distributed file repository (FR). Basically, FR stores files and information about files, and is implemented by servers. Users run clients on their machines and communicate with FR over some network. One server is able support many users. With a suitable front end, users can store and retrieve files. The server maintains state concerning the whereabouts of files; who has copies on which machines, which version is current, and who was granted locks, when locks expire, and so on. Notice, however, that FR does not provide the functionality of traditional source control systems such as RCS or SCCS. FR is meant to provide storage that users utilize from whatever application they might be using. No understanding of any file-formats is built into FR, and the object for replication is whole files. For the same reason, if the logical scope of some datum contains more than one file, it is the responsibility of the client (software run by the user) to operate on all these files in concert.

FR supports replication.[1] Servers will maintain consistency between the replicas by means of a server-to-server protocol [Moxnes, 1997]; this protocol is based on two-phase commit. There are thus two regimes for interaction in the system. The first, labeled "1" in Figure 1(a), is between users and their (local) FR server. The other, labeled "2" in the figure, is the inter-server communication required for replication (usually over WANs). This communication is outside the "reach" of users in the sense that when a file is replicated, the FR will exhibit best effort to keep all replicas consistent.

Furthermore, files can also be *shipped* from one FR server to another (site) by a particular variant of the replication machinery. Shipping files has two applications. First, when the right to access a file is delegated to a user situated at some other site, the file can be shipped to that site. Second,

[1] The term *copy* has to do with distribution in space, while *version* implies changes over time. Replication ensures that all copies of a file has the same version, and a *replica* is a copy which is kept up-to-date.



(a) Two-level replication (b) FR

Figure 1: Two views on FR

before departure to a remote site, a user can delegate to herself in the rôle of visitor at a remote site, access to her own files. In that way, upon arrival, selected files will be present. A typical rationale for shipping files is to ease cooperation at the foreign site, since all necessary files are available when they are needed.

FR is implemented as two distinct parts, the front end (FE) and the back end (BE); details of how FR is structured in shown in Figure 1(b). For each client that connects to FR, a new FE is started. The FE and the client interact by means of a custom designed protocol named FRTP [Stabell-Kulø, 1995]. BE maintain the file store, and communicate with the FEs by means of Sun RPC [Sun Microsystems, Inc, 1988b]. FR is discussed at length in [Stabell-Kulø and Fallmyr, 1998].

2.2 Security in FR

To maintain security, some type of access control is needed. FR uses two of the most widely used security models: access control lists (ACLs) and capabilities. Both models have their merits in that ACLs implement some centralized control and support administrative activities better, while capabilities can be easily forwarded to a sub-server as an authorization to carry out a task [Gong, 1989]. In FR, ACLs are used for local access control, while capabilities are used for remote access control. This paper is only concerned with remote delegations, ie., focus will be on capabilities.

Access to FR is granted to users that can present valid credentials. Or, to be precise, commands are accepted on channels that, by means of credentials, are asserted to speak for a user. Such credentials include a valid certificate issued by the system administrator. Normally, users are repre-

sented by a public key that "speaks for them" [Lampson et al., 1992].

Users may delegate authority to others at will. This paper considers authority in the form of the ability to access a file. How the delegate is to be authenticated is a concern of the delegator, not FR; the mechanisms to ensure this will be discussed. That is, when the owner of a file issues a certificate, he has the sole responsibility for ensuring that the certificate is in accordance with his own security policy. FR merely verifies that the credentials supplied with the certificate is sufficient.

The setting we will consider is one with two users, A and B. They reside on two different sites, possibly separated by a WAN. Both A and B is supported by their respective local FR server S_A and S_B. The problem we will consider is twofold. First, A wants to delegate to B authority to read some file stored at S_A. How can this be accomplished on some extra system channel? Second, revocation must be possible.

In FR, the trusted computing base (TCB) varies in that individual users might in part choose their own. Such a view is consistent with a design strategy placing the user and his equipment in focus. The local FR server must be included in the TCB if FR is to be useful. However, FR is only included in the TCB when we consider the files stored there. Users are free to use whatever client they want. This is in contrast to Kerberos, where there are no such choices, and the users must have the complete system in their TCB (all or nothing).

2.3 Related work

As previously stated, Kerberos puts all trust in a TTP, which we have deliberately avoided in our realization of a personal computing environment.

The delegation model in FR is based on μ-CAP [Helme, 1997], with the main difference being that we extended μ-CAP to support delegations in a multi-domain environment. Also, μ-CAP, which was inspired by ICAP [Gong, 1989], is a capability-only system and thus make no use of ACLs.

DSSA [Gasser et al., 1989] and SESAME [McMahon, 1995] are two systems that were designed to be used in a multi-domain environment and, unlike Kerberos, they both use public-key cryptography and certificate authorities (CAs). They differ in the way they use security servers: DSSA uses off-line security servers; SESAME provides on-line security servers. Some disadvantages of both off-line and on-line security servers are given in [Ashley and Broom, 1997]. Here, some disadvantages of off-line servers seem to be:

- *Management of change*: since there are several copies of certificates, it is difficult to change security information, and revocation is slow.

- *Lack of confidentiality*: user identities and privilege information are readable for all.

And on-line servers seem to have some of the following disadvantages:

- *Availability*: if a server becomes unavailable, the system may come to a complete stop since authentication and access control may not be possible.
- *Compromise*: since on-line servers may store secrets of entities, compromise of the server may be catastrophic.
- *Attacks*: on-line servers may be attacked through the network.

Thus, the main advantage of on-line servers is being able to handle fast revocations, while off-line servers seem more secure in that they are more difficult to attack and compromise.

3 Secure Delegation

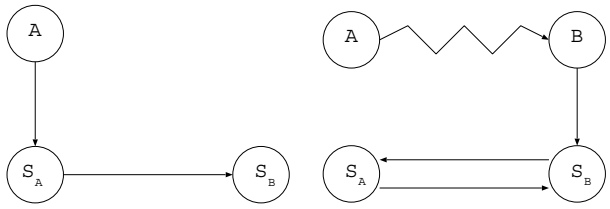
This section is concerned with the design of the delegation mechanism. First, the overall strategy is described, then the representation of users, certificates and messages. We end this section by giving some implementation details, and describe some performance results.

3.1 Setting

The principals involved in a delegation are two users A and B, and both are assumed to be represented by their public key. Being represented by a public key involves a delegation in itself; the details are discussed in Section 4. In our setting, each user has access to a local server, S_A and S_B respective. The common case would be for A and S_A to be situated at one site, while B and S_B would be at another. However, no assumptions are made regarding connectivity.

Assume that A wants to give B access to a file of his. He would construct a certificate, and hand it to B. The certificate includes information about A, B and the file. the certificate is encoded in SDSI [Rivest and Lampson, 1996], and thus in ASCII, and can be sent in an email; the format of certificates is discussed in Section 3.3. B hands the certificate to his server S_B, which will fetch the file from As server S_A.

While designing the delegation protocol, we had two overall modes of operation to choose from, either *push* or *pull*. As seen in Figure 2, a push-protocol has the issuer A pushing the delegated file from the local server to



(a) Push (b) Pull

Figure 2: The push and pull protocols

the remote server. A pull-protocol, on the other hand, requires the delegate B to pull the file, after receiving a delegation certificate from A. Another difference between these two protocols is that the push-protocol requires A to be on-line when issuing the delegation, while A may be off-line with the pull-protocol.

We have chosen, as described earlier, a pull-protocol. A side effect is that an explicit acceptance of the delegation can be implemented. With a push-protocol, a denial of service attack is possible. A file could have been shipped even if it was not wanted, flooding the receiver. We believe the higher complexity of the pull-protocol does not detract from these important factors in our choice.

If the access granted is a *write* access, more protocol machinery is needed in order to ship the file back to the original server. This writeback-protocol is essentially the push-protocol in figure 2, with B initiating the protocol to the local server S_B using the delegation certificate from A. Note that A's local server would now know (because of the delegation certificate) that A expects a file.

If a user wants a delegation certificate revoked, a revoke command can be issued to the local server. Revocation is also provided in that delegation certificates have once-only semantics, which means that a delegation certificate can only be used at-most-once by the acceptor B.

3.2 Protocol description

In this section we will describe the protocol(s) that realize the delegation across domains.

The first message is the delegation certificate, denoted C_a, and is sent from A to B. The goal is to transfer the certificate.

$$\textit{Message 1:} \quad A \rightarrow B: \quad \{A, B, S_B, S_A, FI_a, FA_b, T_a\}_{K_A^{-1}}$$

Here, FI_a is the file identificator (a file name), FA_b is a description of the access rights A want to delegate to B, and T_a is a timestamp generated to make the message unique and thus assure *freshness*; we will return to how this timestamp is used in Section 4. This message is "outside" the rest of the protocol, in that it does not matter how the certificate is transported; it may be through email (encrypted or not), through tele-fax, or even through ordinary mail. This means that FR is not involved when the certificate is actually issued. The important factor is that B is able to represent it digitally before proceeding with the protocol.

When time has come, B initiates actions by asking his local server to retrieve the file.

$$\textit{Message 2:}\quad B \rightarrow S_B :\ \ \{B, S_B, C_a, T_b, N_b\}_{K_B^{-1}}$$

The certificate C_a is handed to S_B in order for S_A to make an access control decision.

On behalf of B the server S_B will contact S_A.

$$\textit{Message 3:}\quad S_B \rightarrow S_A :\ \ \{S_B, S_A, \{B, S_B, C_a, T_b, N_b\}_{K_B^{-1}}, N_{SB}\}_{K_{S_B}^{-1}}$$

The certificate C_a is sent to S_A through S_B. Then, S_A makes an access control decision based on the content of C_a (see Section 4). If authorization is granted, the file is sent to S_B. Notice the implicit delegation to the new principal S_B **for** B. We return to the different principals in Section 4.

If A's server finds that the credentials are sufficient the file is sent to S_B.

$$\textit{Message 4:}\quad S_A \rightarrow S_B :\ \ \{\{S_A, S_B, FI_a, FA_{C_a}, F, N_{SB}\}_{K_{S_A}^{-1}}\}_{K_{S_B}}$$

The file is encrypted to prevent disclosure.

Finally, B is notified that the file has been shipped.

$$\textit{Message 5:}\quad S_B \rightarrow B :\ \ \{S_B, B, FA_{C_a}, H(F), N_b\}_{K_{S_B}^{-1}}$$

The sixth and last message is semantically part of the protocol, even though it is done separately. How B and S_B interacts to transfer the file is outside the scope of the protocol that ensures delegation.

$$\textit{Message 6:}\quad S_B \rightarrow B :\ \ F$$

Although we use certificates with once-read and once-write access, we need to specify a protocol for sending the file F back to S_A. We have specified the following writeback-protocol:

$$\textit{Message 1:}\quad S_B \rightarrow S_A :\ \ \{\{S_B, S_A, C_a, F', N_{B'}\}_{K_{S_B}^{-1}}\}_{K_{S_A}}$$
$$\textit{Message 2:}\quad S_A \rightarrow S_B :\ \ \{S_A, S_B, AC(C_a), N_{B'}\}_{K_{S_A}^{-1}}$$

The first message sends the original certificate issued by A, along with the modified file. The message is encrypted to prevent disclosure. S_A checks the certificate, updates the file (if write access was granted), and replies with $AC(C_a)$ which says if the writeback was accepted or denied. This writeback-protocol occurs by a request from B to S_B which is outside this protocol.

3.3 Implementation

Public keys are used to authenticate users in FR. Key-pairs are created externally with the application Pretty Good Privacy (PGP) [Zimmermann, 1995], and we have written software for using PGP keys in FR to create and verify digital signatures, as well as encrypt and decrypt session keys. We have also written software for handling delegation certificates, using SDSI 1.0 S-expressions. Both our PGP-library and our SDSI-library rely on the crypto-library *SSLeay*[2] for processing encryption (and decryption) tasks, as well as handling large integers.

All keys are stored in the BE of FR; public keys may be transported to a FE by requests where the FE may do verification and encryption operations; the secret key is only handled by the BE who do all signature and decryption operations.

We have means to delegate access rights on files locally, within a domain, and throughout multi-domains; we extended FR with a *change acl* command to delegate access rights locally; within a domain and throughout multi-domains delegation certificates are issued and used. An example of an actual delegation certificate (C_a) is shown in Figure 3. Here, we have used email-addresses to identify the issuer, the acceptor, as well as the servers, but these identifiers may be any PGP UIDs (a name, fingerprint of a public key, etc.). The delegated file is identified as *host.site.domain:/path/to/file*.

FR provides two commands that implements the pull- and writeback-protocols for the acceptor; B may use the *retrieve* command to retrieve the delegated file from S_A; the *writeback* command is used to ship the file back to S_A.

Certificate revocation lists (CRLs) are used for revocation, to prevent replays of once-only delegation certificates, and for revoking existing delegation certificates. To revoke existing certificates, we implemented an *epoch* command which explicit tells FR that all certificates issued before this epoch is no longer valid for this file. Thus, a delegation certificate is only valid within the existing epoch. This *epoch* command also prevents CRLs from

[2]Information about SSLeay can be found at URL http://www.psy.uq.oz.-au/~ftp/Crypto/

```
( Cert:
  ( SDSI-version: 1.0 )
  `( Delegate-To-Client: <ronnya@pasta.cs.uit.no> )
  `( Delegate-To-Server: <fradm@cs.uit.no> )
  `( Delegate-From-Client: <tage@acm.org> )
  `( Delegate-From-Server: <fradm@pasta.cs.uit.no> )
  `( Delegate-File: terje.pasta.cs.uit.no:/test )
  `( Delegate-ACL: read )
  `( Date: 1998-07-20T19:00:26.705+0200 )
  `( Cert-ID: =6b30Rir0uT+YrpB5mG4DNg== )
  ( Signed:
   ( Object-Hash:
    ( MD5 =RDI4NDYyRjgzNENCNDRGMkFBQ0E4MUMzME-
          =IwNjMwMzQ= ) )
   ( Date: 1998-07-20T19:00:26.838+0200 )
   ( Signature: #89009503050035B377AAE5292983107B01D-
                #10101699C0400B9B29E701E3267013A6564-
                #AE4F133ED38AAA36D9A2931607D08037406-
                #8516F14CE5977BA761406F22D8EF9433A15-
                #1237DBBD95C69D45AC17FAE205401886A01-
                #EEF757910152BCD6C2389E64CC89EE2608D-
                #C75B0E0073C5B7054F2809EA88C764DAFFC-
                #C9C6F44432FB23147D154F774A9B83BEDB6-
                #0D51B73E26DA0F8B1CE37DE1 ) ) )
```

Figure 3: Example delegation certificate

	Average	Minimum	Maximum
Insert	14.9	14.1	15.2
Extract	6.5	6.0	8.2

Table 1: Replication performance (two replicas)

	Average	Minimum	Maximum
Pull	18.2	17.4	19.2
Writeback	12.3	12.0	13.4

Table 2: Performance of the pull- and writeback-protocols

growing infinitely.

3.4 Performance

FR has been implemented on FreeBSD, and consists of approximately 25000 lines of source code in C (a third being statements). This does not include the libraries that handles PGP keys, SDSI certificates, and other cryptographic tools.

Our performance test setup consisted of three Hewlett-Packard Kayak workstations running FreeBSD 2.2.7 on a LAN; all workstations had dual PII-300MHz processors and 256 MB RAM each. Two of these workstation were used as servers (running an FR server each); one workstation was used as a client.

We measured the performance of a replicated file, by using the *insert* and *extract* commands, and we also measured the performance of the pull-protocol and writeback. The average times, in seconds, as well as minimum and maximum times are shown in table 1 and table 2. The size of the replicated- and the delegated file were 100 KB.

As seen, *inserting* a file with two replicas used, in average, 14.9 seconds. The *extract* command was substantially faster and spent, in average, 6.5 seconds. The *pull*-protocol, however, used 18.2 seconds in average. The *writeback*-protocol was somewhat faster, and spent, in average, 12.3 seconds.

From this we can tell that the performance of reading- and writing-operations on a replicated file is opposite to the performance of reading and writing with delegation certificates. With a replicated file, writing is slower; with a delegated file, reading is slower. On the surface one might believe that the pull-protocol command would perform better than the *insert* command

(on a file with two replicas), since no file content is sent from the client to the server. However, with the pull-protocol more data are transfered between servers since the file content is base-64 encoded. In addition, the certificate (with the file content) is encrypted. These factors are probably the main reasons for the worse performance results of the pull-protocol (compared to *insert*).

4 Analysis

This section will analyze the protocols previously specified, including analyzing both the delegation and the authentication parts of the pull-protocol, as well as analyzing the authentication parts of the writeback protocol.

4.1 The delegation

Although the BAN logic [Burrows et al., 1990] is widely used for analyzing authentication protocols, it does not say anything about delegations. We can use the theory from [Lampson et al., 1992] for this purpose.

S_A will only surrender a file F belonging to A to a principal B if B can present credentials indicating that A wants the file to be surrendered. That is, B must present to S_A a certificate saying A **says** "B *access* F". We will now show that the pull-protocol gives S_A reason to believe that A intended (for) B to obtain access to the file.

S1: A explicitly mentioned B in the delegation certificate, denoted at Message 1 (see Section 3.2). The interpretation is that when B quotes A he speaks for the compound principal B **for** A.

Formally, A **says** B|A $\Rightarrow$ B **for** A.

S2: By mentioning S_B in his certificate, A acknowledges Bs request for using an intermediate server. If A had left out this principal, B would have had to contact S_A directly. The use of S_B is for convenience only.

Formally, A **says** S_B|(B **for** A) $\Rightarrow$ S_B **for** (B **for** A).

S3: The credentials described above are signed together with the name F of a file. The interpretation is that the new authority granted to B and S_B is limited to this file.

Formally, A **says** "(B **for** A) access F".

S4: When B in Message 2 signs the delegation certificate he accepts the delegation.

Formally, B|A **says** B|A $\Rightarrow$ B **for** A.

S5: The delegation made by A in S2 applies to B only when it is acknowledged. B does that by including the name of S_B in Message 2.

Formally, $(B|A)$ **says** $S_B|(B$ **for** $A) \Rightarrow S_B$ **for** $(B$ **for** $A)$.

S6: Since F is mentioned in the certificate from A and B signs it, he acknowledges that he will not (try to) use the authority for any other purpose.

Formally, $(B|A)$ **says** "$(B$ **for** $A)$ access F''.

S7: In S2 and S5 authority was delegated to S_B, it need to be acknowledged. Message 3 achieves this.

Formally, $(S_B|(B|A))$ **says** $S_B|(B$ **for** $A) \Rightarrow S_B$ **for** $(B$ **for** $A)$.

S8: Finally, S_B uses his new authority (obtained by S6) to ask for the file.

Formally, $(S_B|(B|A))$ **says** "$(B$ **for** $A)$ access F''.

In the analysis we use rules (P1), (D1) and (P11) from [Lampson et al., 1992]. Applied on statements S1 and S4 above, we get:

$$B|A \Rightarrow B \textbf{ for } A \tag{1}$$

Likewise, by applying the same rules on statements S2, S5 and S7, we get:

$$S_B|(B \textbf{ for } A) \Rightarrow S_B \textbf{ for } (B \textbf{ for } A) \tag{2}$$

This means that A has delegated authority to B who has accepted it, and that $(B$ **for** $A)$ has delegated authority to S_B who also has accepted it. Then, when S_A retrieves statements S1-S8, S8 parses out as S6 and S3 because of (1) and (2). To elaborate, statement S8 means that:

$$(S_B \textbf{ for } (B \textbf{ for } A)) \textbf{ says } "(B \textbf{ for } A) \text{ access } F''$$

And (2) means that S_B is able to say so. Then we have:

$$(B \textbf{ for } A) \textbf{ says } "(B \textbf{ for } A) \text{ access } F''$$

And since (1) means that B is able to say so, we end up with:

$$A \textbf{ says } "B \text{ access } F''$$

Which is exactly the information S_A needs in order to grant B (through S_B) access to the file F.

The above analysis shows to which principals authority is delegated, and it shows that the messages support the conclusions. The analysis assumes that all messages are authenticated, and (reasonable) fresh. In the next section, we analyze the protocol to establish whether this is the case.

4.2 The pull-protocol

The previous section verified that the messages include information that enables the participants to conclude that delegation has been properly authorized. However, the analysis assumes that the protocol ensures that all messages are *fresh* and properly authenticated.

In the analysis of authentication, the BAN logic is used, using the notation from [Burrows et al., 1994]. First we specify our goals for the protocol: B should believe that A has delegated him access rights to a file, then B should believe that S_A has granted the delegated access rights, and B should also believe that the delegated file is fresh.

There are three sets of assumptions for the protocol; the first one contains the beliefs the participants have about the encryption keys used throughout the system:

A **believes** $\overset{K_A}{\mapsto} A$ $\quad$ B **believes** $\overset{K_{S_A}}{\mapsto} S_A$ $\quad$ S_A **believes** $\overset{K_{S_A}}{\mapsto} S_A$

A **believes** $\overset{K_{S_A}}{\mapsto} S_A$ $\quad$ B **believes** $\overset{K_{S_B}}{\mapsto} S_B$ $\quad$ S_A **believes** $\overset{K_{S_B}}{\mapsto} S_B$

A **believes** $\overset{K_B}{\mapsto}$ B $\quad$ S_B **believes** $\overset{K_{S_B}}{\mapsto} S_B$ $\quad$ S_A **believes** $\overset{K_A}{\mapsto} A$

A **believes** $\overset{K_{S_B}}{\mapsto} S_B$ $\quad$ S_B **believes** $\overset{K_{S_A}}{\mapsto} S_A$ $\quad$ S_A **believes** $\overset{K_B}{\mapsto}$ B

B **believes** $\overset{K_B}{\mapsto}$ B $\quad$ S_B **believes** $\overset{K_B}{\mapsto}$ B

B **believes** $\overset{K_A}{\mapsto} A$ $\quad$ S_B **believes** $\overset{K_A}{\mapsto} A$

These key beliefs tell that the different principals know their own public keys as well as the public keys of the other principals participating in the protocol. None of them should be controversial.

Assumptions about the *jurisdiction* beliefs:

S_A **believes** A **controls** X_b $\quad$ S_B **believes** A **controls** X_b

B **believes** A **controls** X_b $\quad$ S_B **believes** S_A **controls** X_{S_A}

B **believes** S_B **believes** S_A **controls** X_{S_A}

Here, B, S_B, and S_A believe that A has jurisdiction over the access rights delegated with X_b, and S_B believes that S_A has jurisdiction to construct a response from this delegation. In addition, we have the weak assumption that B believes that S_B believes that S_A has jurisdiction to construct a response from C_a. We need to include this assumption since B will never see a message from S_A, which implies that B trusts S_B when S_B says that there is a response from S_A.

Finally, the last set of assumptions deal with *freshness*:

A **believes fresh**(T_a) $\quad$ B **believes fresh**(N_b) $\quad$ S_B **believes fresh**(N_{SB})

B **believes fresh**(T_a) $\quad$ S_B **believes fresh**(T_b) $\quad$ S_A **believes fresh**(T_a)

Now, idealized the protocol can be specified as:

Message 1: $A \rightarrow B: \quad \{X_b, T_a\}_{K_A^{-1}}$

Message 2: $B \rightarrow S_B: \quad \{\{X_b, T_a\}_{K_A^{-1}} \text{ from } A\}, T_b, N_b\}_{K_B^{-1}}$

Message 3: $S_B \rightarrow S_A: \quad \{\{\{X_b, T_a\}_{K_A^{-1}} \text{ from } A\}, T_b, N_b\}_{K_B^{-1}} \text{ from B}\}, N_{SB}\}_{K_{S_B}^{-1}}$

Message 4: $S_A \rightarrow S_B: \quad \{\{X_{S_A}, F, N_{SB}\}_{K_{S_A}^{-1}}\}_{K_{S_B}}$

Message 5: $S_B \rightarrow B: \quad \{X_{S_A}, H(F), N_b\}_{K_{S_B}^{-1}}$

Message 6: $S_B \rightarrow B: \quad F$

First, B receives Message 1 and from the *public key message-meaning* and *nonce-verification* rules, obtains:

$$B \textbf{ believes } A \textbf{ believes } X_b$$

And from the *jurisdiction* rule, B obtains:

$$B \textbf{ believes } X_b$$

Message 2 is similar to Message 1, where now S_B obtains from the *message-meanings*, *nonce-verification*, and *jurisdiction* rules:

$$S_B \textbf{ believes } B \textbf{ believes } X_b$$
$$S_B \textbf{ believes } A \textbf{ believes } X_b$$
$$S_B \textbf{ believes } X_b$$

From Message 3, S_A obtains (from the same rules as applied above):

$$S_A \textbf{ believes } B \textbf{ believes } X_b$$
$$S_A \textbf{ believes } S_B \textbf{ believes } X_b$$
$$S_A \textbf{ believes } A \textbf{ believes } X_b$$
$$S_A \textbf{ believes } X_b$$

With Message 4 we begin with the reply from S_A, and S_B obtains:

$$S_B \textbf{ believes fresh}(F)$$
$$S_B \textbf{ believes } S_A \textbf{ believes } X_{S_A}$$
$$S_B \textbf{ believes } X_{S_A}$$

B then obtains from Message 5:

$$B \textbf{ believes fresh}(H(F))$$
$$B \textbf{ believes } S_B \textbf{ believes } X_{S_A}$$
$$B \textbf{ believes } S_A \textbf{ believes } X_{S_A}$$
$$B \textbf{ believes } X_{S_A}$$

Finally, with $Message\ 6$ and the *hashing* rule, B obtains:

$$B \textbf{ believes fresh}(F)$$

Which means that we have obtained our goals set for the protocol; B now believes that A has delegated B access rights (X_b) to a file F, that S_A has fulfilled the access rights delegated (X_{S_A}), and that the file F is *fresh*.

4.3 The writeback-protocol

We use the assumptions from the previous section, as well as the additional S_B **believes fresh**(N'_{SB}). Idealized, the protocol can be specified as:

$$\begin{aligned}
&\textit{Message 1:}\quad S_B \rightarrow S_A : \quad \{\{\{X_b, T_a\}_{K_A^{-1}} \text{ from } A\}, F', N'_{SB}\}_{K_{S_B}^{-1}}\}_{K_{S_A}} \\
&\textit{Message 2:}\quad S_A \rightarrow S_B : \quad \{X_{S_A}, N'_{SB}\}_{K_{S_A}^{-1}}
\end{aligned}$$

By applying the *message-meaning*, *nonce-verification*, and *jurisdiction* rules, S_A obtains from $Message\ 1$:

$$\begin{aligned}
&S_A \textbf{ sees } F' \\
&S_A \textbf{ believes } X_b
\end{aligned}$$

And with $Message\ 2$, S_B obtains:

$$S_B \textbf{ believes } X_{S_A}$$

The protocol concludes now that S_B knows, from X_{SA}, that S_A has accepted (or denied) X_b and F'.

4.4 Use of timestamps

Some notes on the use of timestamps in the pull-protocol need to be made. Timestamps were first introduced in [Denning and Sacco, 1981] to prevent replays of compromised keys, as well as replacing a two-step handshake. In the latter case, it is necessary to use synchronized clocks, with the risks that follows [Gong, 1992]. This means that the use of timestamps should be used with care: when is a timestamp *fresh* and are the clocks synchronized? We use timestamps two times in the pull-protocol: in the certificate C_a, and once in $Message\ 2$ ($B \rightarrow S_B$). The use in C_a is prudent, since the timestamp is used to make sure the certificate is not used before (in which we check all previous certificates used). In $Message\ 2$, however, it is used to to tell S_B that the message is *fresh*, which S_B may only verify if the clocks on B's computer and S_B's computer are synchronized.

It should be noted that our assumption B **believes fresh**(T_a) is rather strong. This is because timestamps, in conjunction with freshness, are usually used to indicate that messages are recent. Since both A and B are users, it may take some time between generating C_a to B receives it (this may take hours, even days). Also, since there is no requirement for B to store old certificates, there is no way for B to be absolutely sure that C_a is not used before. A way to make sure that B believes C_a to be fresh, would be for B to send a nonce to A first:

$$\begin{array}{lll} \textit{Message 0:} & B \rightarrow A: & N'_b \\ \textit{Message 1:} & A \rightarrow B: & C_a \end{array}$$

Where C_a includes the nonce N'_b as well. But we still use our initial assumption, B **believes fresh**(T_a), since delegation certificates are usually created by requests from users who want the delegation. Thus, our argument is that B will know that C_a is fresh by verifying that T_a is created after the initial request from B to A. This implies trust in (loosely) synchronized clocks. Also, as stated in [Burrows et al., 1994], what it is about timestamps that causes users to believe they are fresh is outside the province of the BAN logic. Another argument is that even if T_a is not fresh, and B believes it is, it will not have any catastrophic consequences since S_A will (later) verify that T_a is not fresh.

5 Discussion

A concern with the existing pull-protocol, is that there is no way for B to know that the file F (given in *Message 6*) is the file F delegated by A, without trusting S_A to give the proper file to S_B and S_B to give this file to B. This could be solved by putting the hash of the file in the delegation certificate (C_a), but that would require A to be on-line when generating the certificate. The reason for this is that if A is off-line, he can not be certain that F has not been changed since the last time he was on-line.

The use of certificates with once-only semantics has the implication that a delegation certificate can not be replayed. This seems better at preventing misuse than the *guaranteed phase* described in [Rivest, 1998], where a certificate is definitely good until the expiration date. The trade-off is the overhead for users since new delegation certificates need to be generated every time a delegation is needed. But we believe that generating certificates is cheap compared to the potential misuse of such certificates. The other reason for using certificates with once-only semantics is that FR was designed to support users who may be regularly disconnected and thus have no means to issue an immediate revocation order to FR [Helme, 1997].

The once-only semantics assures that a delegation certificate will be used at-most-once. Note that the notion of once-only semantics is not completely new; it is said that using electronic money—e.g., spending it when running a pay-per-use program—is analogous to the revocation of a capability [Yee, 1994].

FR also supports chain of delegations, where a delegated file may be delegated, by the original acceptor, to another user again. A flag could have been included in the delegation certificate that said if A would like the file F to be included in such a chain of delegations. However, FR has no means to prevent this from happen outside the system since B could, if necessary, just e-mail F to another user. Also, since F has been shipped to B, B is now the *owner* of the shipped file and one could argue that how this file is further handled is a choice that only B should make.

6 Conclusion

We have implemented a flexible authorization and access control infrastructure for FR, where delegations might take place outside the system, by any means available to the users. Local access control is maintained by use of ACLs; remote access control is done by using capabilities that are valid at-most-once.

There may be occasions where a user wants to share a file and expects a high amount of writes by the other user. Issuing delegation certificates for each write may be a tad cumbersome in this case, especially if the issuer trusts this acceptor more than other users. A more flexible solution could be to use the *guaranteed* phase in [Rivest, 1998] instead, so that the delegation certificate would be valid until a specified expiration date. As discussed earlier, the trade-off is between *convenience* (no need to generate new certificates) and *security* (potential misuse).

Currently, FR make no use of any security servers; delegation- and identity-certificates propagate off-line, outside FR. This means that FR does not scale well. Work is in progress on designing and implementing a public key infrastructure (PKI) which will make use of on-line security servers.

Acknowledgements

We are grateful to past and current collaborators in the Pasta, MobyDick, and GDD projects. In particular, Gunnar Hartvigsen provided valuable input on an earlier version of this paper.

Availability

The FR server can be downloaded from its home page, found at:

`http://www.pasta.cs.uit.no/frserver/`

References

[Ashley and Broom, 1997] Ashley, P. and Broom, B. (1997). A Survey of Secure Multi-Domain Distributed Architectures. Technical Report FIT-TR-97-08, Faculty of Information Technology, Queensland University of Technology, Australia.

[Burrows et al., 1990] Burrows, M., Abadi, M., and Needham, R. (1990). A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36.

[Burrows et al., 1994] Burrows, M., Abadi, M., and Needham, R. (1994). The Scope of a Logic of Authentication. Technical Report 39 (Appendix), Systems Research Center, Digital Equipment Corporation. SRC Research Report 39 was originally published on February 28, 1989, and revised on February 22, 1990. This is an appendix to the revised version.

[Denning and Sacco, 1981] Denning, D. E. and Sacco, G. M. (1981). Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24(8):533–536.

[Gasser et al., 1989] Gasser, M., Goldstein, A., Kaufman, C., and Lampson, B. (1989). The Digital Distributed System Security Architecture. In *Proceedings of the 12th National Computer Security Conference*, pages 305–319, Baltimore, MD.

[Gong, 1989] Gong, L. (1989). A secure identity-based capability system. In *IEEE Symposium on Security and Privacy*, pages 56–63, Oakland, California, May 1989. IEEE.

[Gong, 1992] Gong, L. (1992). A security risk of depending on synchronized clocks. *ACM Operating Systems Review*, 26(1):49–53.

[Helme, 1997] Helme, A. (1997). *A System for Secure User-controlled Electronic Transactions*. PhD thesis, University of Twente. ISBN 90-423-0011-6.

[Kohl and Neuman, 1993] Kohl, J. and Neuman, B. (1993). RFC 1510: The Kerberos Network Authentication Service (V5).

[Lampson et al., 1992] Lampson, B., Abadi, M., Burrows, M., and Wobber, E. (1992). Authentication in Distributed Systems: Theory and Practice. *ACM Transactions on Computer Systems*, 10(4):265–310.

[McMahon, 1995] McMahon, P. V. (1995). SESAME V2 Public Key and Authorization Extenstions to Kerberos. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, pages 114–131. IEEE Computer Society Press.

[Moxnes, 1997] Moxnes, G. (1997). Design og implementasjon av replikering i File Repository (in Norwegian). Master's thesis, University of Tromsø.

[Rivest, 1998] Rivest, R. L. (1998). Can We Eliminate Certificate Revocation Lists? In *Proceedings of the Second International Conference on Financial Cryptography (FC'98)*, Anguilla, British West Indies.

[Rivest and Lampson, 1996] Rivest, R. L. and Lampson, B. (1996). SDSI – A Simple Distributed Security Infrastructure. Available at http://theory.lcs.mit.edu/~cis/sdsi.html.

[Stabell-Kulø and Fallmyr, 1998] Stabell-Kulø, T. and Fallmyr, T. (1998). User controlled sharing in a variable connected distributed system. In *Proceedings of the seventh IEEE international Workshop on Enabeling Technologies: Infrastructure for Collaborative Enteprises (WET-ICE'98)*, pages 250–255, Stanford, California, USA. IEEE Computer Society, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitor, CA 90720-1314, USA.

[Stabell-Kulø, 1995] Stabell-Kulø, T. (1995). File Repository Transfer Protocol (FRTP). Technical Report CS-TR 95-21, Department of Computer Science, University of Tromsø, Norway. Available as http://www.cs.uit.no/forskning/rapporter/Reports/9521.html.

[Sun Microsystems, Inc, 1988a] Sun Microsystems, Inc (1988a). RFC 1050: RPC: Remote Procedure Call Protocol specification. Obsoleted by RFC1057 [Sun Microsystems, Inc, 1988b].

[Sun Microsystems, Inc, 1988b] Sun Microsystems, Inc (1988b). RFC 1057: RPC: Remote Procedure Call Protocol specification version 2. Obsoletes RFC1050 [Sun Microsystems, Inc, 1988a].

[Yee, 1994] Yee, B. S. (1994). *Using Secure Coprocessors*. PhD thesis, Carnegie Mellon University.

[Zimmermann, 1995] Zimmermann, P. (1995). *The Official PGP User's Guide*. The MIT Press, Cambridge, Massachusetts, 02142 USA.