WDAI: a simple World Wide Web distributed authorization infrastructure

José Kahan

W3C/INRIA, 655, av. de l’Europe, 38330 Montbonnot St.-Martin, France

Abstract

The World Wide Web (W3) has the potential to link different kinds of documents into hypertext collections and to distribute such collections among many document servers. Distributed collections can bring forth new W3 applications in extranets and expand the concept of content reuse. However, they also bring new authorization problems, such as the need for coordinated user administration, user authentication, and revocation of rights. This paper proposes WDAI, a simple and general infrastructure for distributed authorization on the World Wide Web. Under WDAI, browsers and servers exchange authorization information using X.509v3-based authorization certificates. WDAI is designed to be open to a wide variety of security policies and, being compatible with existing W3 technology, can be implemented without modifying existing browsers.

Keywords: World Wide Web; Distributed hypermedia; Security; Access control; Distributed authorization

1. Introduction

The World Wide Web (W3) provides the means to link different kinds of documents into hypertext collections. Most of today’s web sites centralize the documents that form a collection in a single location. However, it is possible to create distributed collections by linking documents which are stored in different servers.

Distributed hypertext collections bring the advantage of being able to store parts of a collection in servers which are closer to authors. For example, in an extranet made of different research centers, we can store the documents generated by each research center in their own local servers, without precluding the linking of a document to any other one in the extranet.

The World Wide Web Consortium SMIL Recommendation [22] offers another potential application for distributed collections. For example, a SMIL document can be stored in one server and describe the presentation of multimedia contents stored in other servers. The contents could be produced by other authors and be leased to the SMIL author.

Unfortunately, distributed hypermedia collections not only bring benefits, but also new authorization problems. It is necessary to coordinate user administration, specification of access rights, and revocation of authorization for the documents that belong to a collection [11,12]. Different research and commercial techniques have been proposed to solve some of these problems [2,8,14]. However, these solutions are not well adapted for distributed collections, im-
pose the use of an enhanced browser, impose the use of an intermediary proxy or gateway, or impose a security policy.

In this paper, we propose WDAI, a simple W3 distributed authorization infrastructure for collection of documents that may be stored in different servers. The main features of WDAI are as follows (not given in any particular order):

- **Standard hypertext navigation.** Users can bookmark protected documents and jump from one document to another, without having to follow all the in-between links.
- **Support for document sharing.** Documents can be included in collections belonging to different authorization domains.
- **Security policy independency.** WDAI does not impose any security policy. As security needs vary from system to system, they cannot be generalized to all systems. Instead, WDAI provides the means for expressing a wide variety of security policies.
- **Server independence.** To avoid a potential denial of service and to improve response time, the document servers take their access control decisions locally, without having to systematically consult other servers.
- **W3 compatibility.** The authorization protocols proposed by our infrastructure work directly over HTTP and HTTPS. We don’t introduce new protocol headers and we respect the standard stateless HTTP protocol. Moreover, WDAI supports off-the-shelf SSL-enabled browsers.

The next section presents the WDAI design. Section 3 gives an informal security analysis of WDAI. Section 4 describes five security policies and shows how they can be interpreted under WDAI. There are other techniques that partially address the needs for distributed W3 authorization; some of them are discussed in Section 5. Section 6 concludes the paper and presents some of the perspectives for future work on WDAI.

### 2.1. Assumptions

Since the inception of W3, a user may, technically, make a uni-directional link to any other document in any other server [4]. The W3 philosophy makes it the responsibility of the owner of a document to establish access rights to the document. However, when trying to control access to a distributed collection of documents, it is necessary to coordinate the authorization information used to describe and to evaluate the access rights of users. We represent this need for coordination by grouping collections into authorization domains. An authorization domain may comprise one or more collections, but a collection may not belong to more than one domain. Documents may belong to more than one collection even if those collections belong to different authorization domains.

In general, when a server tries to authorize a request for its resources, the server needs to evaluate the request against the authorization information which specifies the access rights of the user [21]. In WDAI, part of this information is local to each server and part of it is coded inside an X.509v3 [10] based authorization certificate that the browser includes in each request. Because of certificate revocation concerns, our infrastructure assumes that there is a loosely secure synchronized clock in the authorization domain. This problem is similar to the one faced by Kerberos based systems [18]. [9,13] propose some solutions on how to achieve secure clock synchronization.

Distributed collections also introduce the problem of user administration [2,12,11]. The security administrator must provide updated authentication and access control information to all the document servers in its domain. WDAI simplifies user authentication as only the authorization server needs to share authentication information with the users. However, in order to make document servers independent of other servers, each server needs to have a local copy of the access control information specifying the rights of users over its documents. User administration can be simplified by using security policies which are not based on user identity. These policies and the distribution of access control information are current research topics. For the purpose of this paper, we assume that this information is already available at the document servers.
A user should be able to use any standard browser he wants to consult a collection. This includes the case where the user goes to a cybercafé, thus precluding the download of plug-ins or other special programs. Our solution is to use the authorization certificate both to exchange authorization and authentication information. For the browser, there’s no difference between an authorization certificate and a public-key certificate. The browser should be capable of running HTTP over a secure protocol, such as SSL, which guarantees both authentication and confidentiality. In addition, the browser must be able to download X.509 certificates and be able to present them to a server. Both of these features are supported by current SSL-enabled browsers and don’t require a new public-key infrastructure. The details of SSL are not important for the infrastructure, so SSL could be replaced by other protocols.

The browser should be trustworthy while it is being used [1]. It is difficult to control what a browser or a user can do with the browsed documents. In order to avoid overloading the infrastructure with specific precautions against those problems, we assume some trust in the browser. This implies that the browser doesn’t have security faults and that the system in which it is running is under the control of the user. It also implies that we trust authorized users.

2.2. WDAI and its operation

In WDAI, document collections are grouped into authorization domains. In addition, an authorization domain comprises a number of document servers, an authorization server, and a security administrator (Fig. 1).

In an authorization domain, we distinguish two kinds of authorization: a global authorization enforced by the authorization server and a local authorization enforced by each document server.

The authorization server is responsible for the global authorization to the collections in the domain. It is also responsible for the global authentication of users to the domain. The server will grant users authorization certificates according to the information stored in an authorization database. An authorization certificate is an X.509 certificate which includes not only a public-key, but also authorization information.
attributes. Authorization certificates are exchanged between browsers and document servers during the authentication protocols.

Document servers are responsible for the local authorization to the documents that they store. We represent the security policy which governs the access to each document by means of an extended access control list (EACL). When receiving a request, the server first authenticates the user by means of the authorization certificate. The server then determines the user’s access rights by evaluating the authorization attributes stored inside the certificate together with the EACL. In order to validate an authorization certificate, a document server needs to know the public-key of the authorization server.

The security administrator is a conceptual entity. It represents the need for coordinated administration of the information stored in the authorization database and in the EACLs.

The following scenario explains the interaction between these components. Suppose that our distributed collection represents an electronic library. The security policy of the domain allows an authorized user to consult all the documents in the collection. For this example, we consider that every authorized user has a password protected account on the authorization server. To consult the library, user Bob proceeds as follows:

- Bob finds a computer connected to the Internet. The terminal must be running a browser capable of doing HTTPS and allow the download of X.509 certificates.
- Bob browses the library’s home page and gets the URL of the library’s authorization server. Alternatively, Bob could try to browse one of the document’s belonging to the library and get back a message describing the collections to which the document belongs and a list of authorization servers that can grant an authorization certificate.
- Bob contacts the authorization server and retrieves the server’s public-key certificate. Bob then retrieves a form for requesting an authorization certificate for consulting the library’s documents. The form allows Bob to enter his login name and password. The form also instructs Bob’s browser to generate a session public-key pair and to send back the public-key inside a certificate request message [19]. Bob then opens an HTTPS channel with the authorization server and sends back his filled-in request.
- The authorization server consults its authorization database to authenticate Bob and to authorize his request. The server then generates an authorization certificate by concatenating some authorization attributes and an expiration date to the browser’s public-key. Finally, the server signs the authorization certificate and sends it back to the browser.
- Bob instructs the browser to install the authorization certificate and to use it as Bob’s default public-key certificate.
- When Bob tries to access one of the document servers of the library, the server will request the browser to send it the authorization certificate as part of the SSL authentication protocol. Once both parties have been authenticated, the server will use the authorization certificate and the document’s EACL to decide if Bob has enough rights to access the document. The server will then send back the document.
- When Bob finishes his session, he’ll delete the authorization certificate from the browser.

In the above protocol, we described several cryptographic operations on the client side: the downloading of public-key certificates, the generation of a key-pair, the selection of a default user public-key certificate, and the removal of a certificate. All of these operations are supported by today’s most popular SSL-enabled browsers [16,17].

Note that the authorization server also acts as a certification authority [10]. In the above example, the authorization server shares a secret (the password) with the user. After having authenticated the user and authorized his request, the server will grant him an authorization certificate. Such certificate contains a public-key that the document servers should use to authenticate the user. This mechanism eases user administration as the document servers don’t need to share any authentication information with the users; they just need to know the public-key certificate of the authorization server to be able to validate the authorization certificates.

WDAI doesn’t impose any policy on what requirements the user must satisfy in order to acquire an authorization certificate. Our example describes a policy where the user must have some kind of acc-
count on the authorization server. In another policy, the user could have presented a public-key certificate to the authorization server, instead of a password. Still another policy could be one where the user sends a credit-card number instead of a password and buys a time-limited certificate.

We will now describe more in detail the structure of authorization certificates, of the EACLs, and the revocation of such certificates. Appendix A gives a more formal description of the authorization protocols that were mentioned in the scenario.

2.3. Authorization certificates

WDAI uses authorization certificates to exchange authorization information between browsers and servers. These certificates are generated by the authorization server.

The authorization certificates are based on the latest version (version 3) of the X.509 certificates [10]. X.509v3 certificates permit the integration of user defined data, so called extensions, into the certificate. This property, which allows security administrators to bind arbitrary attributes to a public-key, makes certificates more flexible, bringing them closer to the concept of ECMA's Privilege Attribute Certificates [5,6].

Table 1 summarizes the attributes which characterize an authorization certificate. The certificate comprises four kinds of attributes: authentication attributes, authorization attributes, validation attributes, and other standard X.509 attributes (not shown in the table).

The authentication and validation attributes are those defined in X.509. The former provide the public-key of the user. The latter protect the certificate against forgery and specify the certificate’s validation period. While X.509 certificates is issued by a certification authority and is expected to have a relatively long life, an authorization certificate is issued only by the authorization server and has a short life.

The authorization attributes constitute our principal extension to X.509. The collection unique identifier and the collection instance counter attributes identify an instance of a collection in a domain. For example, the collection unique identifier could be made of the domain’s name, a user-readable name, and a binary identifier. The counter allows to distinguish between instances of a collection and is used for revocating active certificates. We’ll describe the revocation of certificates later on.

The other authorization attributes give the privileges of the user, such as user id, groups, roles, security level, and so on, and they may complement the control attributes of EACLs. The value and type of these privileges depend on the security policy of the domain and are not defined by WDAI. Section 4 gives some example on how privileges can be used to implement different security policies.

2.4. Extended Access Control Lists (EACLs)

EACLs specify the access rights of users over the documents stored in a server. Each document is bound with an EACL. As Table 2 shows, each entry in the EACL comprises both authorization and validation attributes.

Table 2
An extended access control list entry

<table>
<thead>
<tr>
<th>Authorization attributes</th>
<th>Control attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorized HTTP request methods</td>
<td></td>
</tr>
<tr>
<td>unique identifier</td>
<td></td>
</tr>
<tr>
<td>Collection instance counter</td>
<td></td>
</tr>
<tr>
<td>Unique identifier of the authorization server</td>
<td></td>
</tr>
<tr>
<td>Unique identifier of the authorization server's public-key certificate</td>
<td></td>
</tr>
<tr>
<td>Signature of the certificate by the authorization server</td>
<td></td>
</tr>
</tbody>
</table>
The authorization attributes describe the access rights of users over the document. We first have the valid HTTP request methods that a user may make on the document. Then, according to the security policy, we may have one or more control attributes that complement the privilege attributes found in the authorization certificate; for example, user id, groups, roles, and so on.

The validation attributes are used to authenticate the certificate itself. They specify both the collection instance in which the authorization certificate is valid and the current public-key of the authorization server that can issue such certificate.

Note that each EACL entry identifies both a collection and an authorization domain. This makes it possible to share documents among different collections and domains.

2.5. Revocation of authorization certificates

The previous sections describe how a user obtains an authorization certificate and how he uses it to request a document. This section briefly examines the revocation of such certificate.

The inclusion of a validity period in the authorization certificates provides an implicit revocation of the certificates. However, in some cases, one may need to make an explicit revocation of active certificates; for example, when public-keys are compromised, when the security of a user or an authorization server is compromised, or when the security state of a document or of a collection of documents evolves. In these examples, the revocation can have different ranges:

- revocation of all the certificates held by a user,
- forbidding the access to a document to users with active authorization certificates,
- revocation of all the certificates of a given authorization domain that give access to a document collection,
- and revocation of all the certificates granted by an authorization server.

We’ll now briefly describe how to take into account all of the above by means of the validation attributes stored in the authorization certificate and in the EACL. For simplicity, we’ll consider that there is only one authorization domain. If a revocation operation spans multiple authorization domains, the operation should be repeated for each concerned domain.

- Revocation of all the certificates held by a user. The security administrator must send the serial numbers of the invalid authorization certificates to all the document servers in its domain.
- Forbid the access to a document in a collection. We comply with this operation by means of the collection counter attribute. Valid authorization certificates need to have a collection counter value equal or bigger to that of the EACL entry. By updating the EACL entry, we can then forbid the use of existing certificates to access the document. The security administrator must coordinate this operation with the authorization server, to insure that new certificates will have a valid counter value.
- Revocation of all the certificates that give access to a collection. This operation is the same as the one described in the previous item, but applied to all the documents that belong to the collection.
- Revocation of all the certificates issued by an authorization server. This requires changing or suppressing the public-key certificate of the authorization server and informing all the document servers in the domain.

In theory, all of the above revocation techniques can be supported in WDAI; however, in practice, their implementation presents some problems. The first technique requires that all servers keep a copy of the invalid certificates until their expiration. All but the second technique require that the security administrator propagates some security information to all the concerned document servers. The fourth technique is the most interesting of all as it allows the protection of documents against the compromise of the authorization server. It can be implemented by storing the public-key certificates of authorization servers in directory servers, and then having each document server periodically retrieve the public-key certificates of the authorization servers which are cited in its EACLs. The retrieval period must be fixed in function of the number of public-keys that must be downloaded, the number of document servers in the domain, and the trust required by the security policy. A document server can also detect when an authorization server’s...
Table 3
Effect of the compromise of WDAI’s elements

<table>
<thead>
<tr>
<th>Compromised element</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorization certificate (theft)</td>
<td>Without effect, countered by user authentication</td>
</tr>
<tr>
<td>Authorization certificate’s private-key (browser or user compromise)</td>
<td>Compromise of a collection</td>
</tr>
<tr>
<td>Document server</td>
<td>Compromise of all the documents in the server</td>
</tr>
<tr>
<td>Authorization server’s private-key</td>
<td>Compromise of all the documents in the domain</td>
</tr>
</tbody>
</table>

public-key certificate has changed by means of the validity period attribute given in the authorization certificate.

3. Informal security analysis

The previous sections describe the design of WDAI. This section analyses the security properties that it achieves. The main theme of these properties is how well WDAI can take the compromise of its elements. The authentication protocol is responsible for protection against message replay, so it’s not included in our analysis. Table 3 gives the summary of our analysis.

As noted earlier, the only protection that WDAI offers for the compromise of an authorization certificate is its implicit or explicit revocation. The former is costly in time, the latter is costly in complexity and resources.

Note that the the compromise of the private-key of the authorization server has the strongest impact on the system. The recovery of such compromise requires changing or deleting the server’s public-key certificate. The speed of recovery depends on how often the document servers refresh their copy of this certificate and whether an attacker can block that refresh.

Table 4
Sample web security policies

<table>
<thead>
<tr>
<th>Security policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-or-nothing</td>
<td>Authorized users may access all the documents in the authorization domain.</td>
</tr>
<tr>
<td>Identity-based</td>
<td>Each document must state which users can access it.</td>
</tr>
<tr>
<td>Role-, group-based</td>
<td>A user belongs to or can assume one or more user groups/role-names. Documents state which groups/roles users must belong to/assume to access them.</td>
</tr>
<tr>
<td>Multi-level security</td>
<td>All documents have a security classification and all users have a security level. Users may not access documents having a higher classification rate than their own security level.</td>
</tr>
<tr>
<td>Collection-based</td>
<td>All the documents are grouped into collections and the granularity of protection is a collection.</td>
</tr>
</tbody>
</table>

4. Security policy examples

In this section, we describe how the security infrastructure can be used to implement five different security policies. The first three policies, which we name all-or-nothing, identity-based, and group-/role-based [2], can be found today in many web sites. The last two, multi-level security and collection-based [11,12], are examples of two potential policies. Table 4 gives an informal description of these policies.

Interpreting a security policy under WDAI is a two step procedure, which involves the generation of an authorization certificate and the selection of security attributes that will be stored both in the certificate and in the EACLs. The authorization server makes the first decision on whether to give an authorization certificate to the user and, if this is the case, which privilege attributes should be included in the certificate. This procedure depends on both the WDAI implementation and the security policy of the domain.

The selection of security attributes follows closely the description of the security policy. Table 5 shows the privilege and control attributes that could be used to implement the policies. Note that the all-or-nothing policy doesn’t use any security attributes, as it only requires the presentation of a valid authorization certificate.
Table 5
A WDAI interpretation of the policies given in Table 4

<table>
<thead>
<tr>
<th>Security policy</th>
<th>Authorization certificate</th>
<th>EACL</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-or-nothing</td>
<td>nothing</td>
<td>nothing</td>
</tr>
<tr>
<td>Identity-based</td>
<td>user id</td>
<td>user id</td>
</tr>
<tr>
<td>Role-, group-based</td>
<td>role name/ role name/</td>
<td>group name/grou</td>
</tr>
<tr>
<td>Multi-level security</td>
<td>security level/</td>
<td>security classification/</td>
</tr>
<tr>
<td>Collection-based</td>
<td>collection name/</td>
<td>collection name/</td>
</tr>
</tbody>
</table>

In some cases, a domain may have multiple security policies. For example, one document server may follow the all-or-nothing security policy and another an identity-based policy.

Rather than having a user request two different authorization certificates, the authorization server can include all the security privileges of the user in a single certificate. Document servers can then retrieve from the certificate the attributes they need to authorize a request. Some convention should exist in the domain in the way of describing such attributes in the certificate, e.g., by using an XML DTD, so that document servers can find and interpret them easily.

The ECMA-138 standard [5] and the Generic Authorization and Access Control API (GAA-AAPI) [20] propose an extensive list of security attributes which may be used to implement these and other security policies.

5. Related work

The RBAC/Web system [2,3], developed by NIST, is an implementation of role-based access control for W3. The system comprises document servers which enforce RBAC at the granularity of individual URLs, a centralized authorization database, and an administrative tool that allows a security administrator to define roles, inter-role relationships, and assign roles to users. A user starts a browsing session by first contacting a session manager service to select an active role. When the user requests a document, the document server uses a secure channel to forward the user’s request and authentication information to an RBAC/Web authorization server.

Upon reception of the authorization decision, the document will either return the document or an error message to the user.

Being based on a centralized authorization server, RBAC/Web has simple user administration needs and supports the instant revocation of access rights. However, there’s a strong dependency on the availability of the server and each request for a document implies an additional request to the authorization server. In WDAI, once the user obtains an authorization certificate, there is no further need to contact the authorization server during the session. However, the explicit revocation of active authorization certificates can be complex. Also, user administration can be complex as there’s a need to synchronize the EACLs of the documents that belong to a collection. User administration can be simplified by using non-nominate security policies [2,12].

Both OSF’s DCE-Web [14,15] and SSE’s TrustedWeb [23] are systems which can be used to support distributed authorization in a W3 environment. WDAI and both these systems work under a similar principle: a user first obtains a certificate from a security server. The certificate includes both authentication and authorization information. From then on, the user presents the certificate to access documents stored in other servers. The document servers maintain their own access control lists.

DCE-WEB and TrustedWeb provide mature interfaces for administrating the user’s access rights. Both systems support a wide variety of security policies, including role-based ones. However, in both systems, the handling of certificates requires the installation of a custom proxy in the user’s workstation, or, in the case of DCE-WEB, the use of an enhanced browser or of a specialized gateway. In WDAI, authorization certificates are based on X.509v3 certificates and can be handled by any standard SSL-enabled browser. Thus, there’s no need to enhance browsers or to use a custom proxy. WDAI doesn’t yet specify any administration interface.

6. Conclusions and future work

W3 has the potential to distribute collections of documents over a number of servers. WDAI provides a simple infrastructure for controlling access to such
collections. The principal contributions of WDAI as follows:

- WDAI doesn’t impose any security policy. Rather, it invites security administrators to define their policies in terms of security attributes and access control rules. WDAI specifies the infrastructure allowing administrators to grant a set of privilege attributes to a user as well as the protocols needed to exchange those attributes between users and servers.
- Use of an authorization server. Security administrators are free to select both the initial user authentication protocol (e.g., password, IP address, public-keys) and authorization requirements (e.g., having an account, sending a credit-card number) for obtaining an authorization certificate. Afterwards, user authentication and authorization will depend on the data stored inside the authorization certificate.
- Simplified user administration. Document servers only need to know the public-key of the authorization server; there is no need for them to share authentication secrets with the users.
- Handling of authorization certificates. By embodying the authorization certificates as X.509v3 extensions, we are able to exchange such a certificate during the SSL authentication protocol and download them into standard SSL-enabled browsers. The details of SSL are not important for WDAI, so SSL could be replaced with other protocols.

Finally, we believe that the most important contribution is to provide a simple infrastructure for experimenting and developing new W3 distributed authorization technologies. Our wish list for future work on WDAI includes:

- delegation of authorization certificates between users,
- a language for expressing access control rules in EACLs and security attributes in authorization certificates,
- collection-based access control,
- sharing of protected documents among collections,
- remote administration of access rights,
- and your own contributions to the field!

Note that existing distributed systems technology can answer many of the above points. When possible, future work on WDAI should try to integrate that technology into a W3 context.

We are currently building a prototype of WDAI, code-named Tartu, around the Apache HTTP server and mod_ssl [7] (support of HTTPS inside Apache). We have working authorization and document servers and have been successful in generating the authorization certificates and using them from SSL-enabled browsers. We are now working on an Apache authorization module for providing customizable access control rules. Tartu will be distributed as Open Source.

Acknowledgements

I thank the reviewers, Ralph Swick, and Frédéric Séraphine for their valuable comments and recommendations. I also thank Gareth Rees and Michael McClennen for providing useful insight which allowed me to generalize WDAI’s design.

Appendix A. WDAI authorization protocols

WDAI defines two authorization protocols, the request for an authorization certificate protocol and the request for a document protocol. Both of these protocols are built over HTTP and HTTPS and use the standard HTTP methods.

In order to simplify the description of the protocols, we’ll assume that the authentication of a principal and the setup of a secure channel take place immediately after the exchange of the principal’s public-key certificate. Table A.1 gives the notation used to describe the protocols.

A.1. Request for an authorization certificate

A user begins a session by first obtaining an authorization certificate from the authorization server. Fig. A.1 summarizes the message exchanges.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → B : M</td>
<td>Principal A sends a message M to principal B</td>
</tr>
<tr>
<td>M1, M2</td>
<td>Concatenation of two messages</td>
</tr>
<tr>
<td>CKU_A</td>
<td>The public-key certificate of principal A</td>
</tr>
<tr>
<td>AuthorCert_A</td>
<td>The authorization certificate of principal A</td>
</tr>
<tr>
<td>M!</td>
<td>A request for message M</td>
</tr>
</tbody>
</table>
authentication of the authorization server

1. Browser → Authorization Server : CKU

2. Authorization Server → Browser : CKU

request for the authorization certificate

3. Browser → Authorization Server : AuthorCert

authorization of the request


5. Browser → Authorization Server : Authorization Information

6. Authorization Server → Browser : AuthorCert

or

Error message

Fig. A.1. Protocol for requesting an authorization certificate.

WDAI doesn’t define the authorization information of Steps 4 and 5; it’s up to the security administrator to define its type and value. For example, it could be a login/password pair, a public-key, a credit-card number, and so on.

Note that only the authorization server needs to be aware of this authorization information. The authorization certificate will provide the authorization and authentication information needed by the document browsers.

A.2. Request for a protected document

Once the user has obtained an authorization certificate, he may start consulting the documents of the collection. Fig. A.2 shows the message exchanges.

Although the protocol seems quite complex, in practice, it’s implemented as a single URL. For example, in Tartu, our prototype implementation of WDAI, URLs for protected documents have the following form:


In the above example, the authorization certificate gets exchanged during the SSL authentication protocol. The server will first authenticate the user using the public-key stored in the certificate. It will then authorize the user using the authorization information provided by the same certificate.

The error message can give be a classic 40x HTTP message. Following the security policy, the message could tell the user what kind of authorization certificate he needs, give the list of all the collections where the document is included, and give the URL of the authorization servers which can deliver the authorization certificates. This information is already present in the document’s extended access control list.

mutual authentication

1. Browser → Document Server : CKU

2. Document Server → Browser : CKU

Document Server

request for the document

3. Browser → Document Server : AuthorCert

authorization of the request


or

Error message

Fig. A.2. Protocol for requesting a document.
References


José Kahan joined the technical staff of the World Wide Web Consortium (W3C), at INRIA Grenoble in January 1996. He participates in the development of Amaya and various other projects. His research interests include distributed systems, W3 security, and hypertext mailing list archives. José holds a Ph.D. in computer science from the Université de Rennes I (1997) and a specialization degree in computer networks from the École Supérieure d’Électricité (SUPELEC), Rennes.