Authentication and Access Delegation with User-Released Certificates

Lavinia Egidi
Dipartimento di Informatica
Univ. del Piemonte Orientale
Spalto Marengo 33
15100 Alessandria, Italy
lavinia@mfn.unipmn.it

Maurizio Melato
NICE srl
Via Serra 33
14020 Camerano Casasco, Asti, Italy
maurizio@nice-italy.com

ABSTRACT

We propose an authentication and access delegation system based on an unconventional use of X.509 certificates. It allows users to connect from any untrusted machine and to define dynamically a group of trusted co-workers. It is low cost, doesn't need unusual software nor hardware on the client's side, and offers a good degree of security without requiring that the user be too careful. The underlying idea is to enable users to release their own certificates with very short life span (or usable just once) to authenticate themselves to the server.

Keywords

Authentication, access delegation, user-released certificates, short-lived and one-time certificates.

1. INTRODUCTION

We propose a short-lived password authentication system based on public-key certificates that allows for user-defined delegation of access. It permits a high degree of user mobility and is low-cost.

Our aim in designing the protocol was to offer flexible authentication mechanisms, while preserving acceptable security levels, using off-the-shelf browsers and a commonly used server. We specifically had in mind a highly mobile user, not very security conscious, that should not be charged with much responsibility, and that needs dynamical sharing of data. In other words (a) our user can need to access sensible information on a web-server from any (untrusted) machine; (b) our user needs to share documents and information with selected co-workers; (c) the sharing groups change in time and must be managed in a dynamical way; (d) we cannot depend upon the user to follow a strict security protocol; (e) we do not wish to equip the user with expensive hardware; (f) we must assume very ordinary hardware and software on the remote host; (g) we still wish to maintain a reasonable level of security.

The prototypical user we initially had in mind is a student at our University; most of our students live in their own home towns, in an area of up to 30-50 km from the University buildings. The University buildings are in three different towns and not even grouped in campuses.

Although our situation is somewhat extreme, the system we propose has features that can be interesting for low-budget companies, for non critical or non educated staff of large companies, for flexible access management in large structures.

The underlying idea is to release to end-users certification authorities (CA), in order to enable them to create their own certificates and, in turn, CAs. The certificates signed by user-CAs have very short time validity, thus allowing for careless use in non trusted environments. Releasing a subordinate CA to another user, and properly setting a configuration file on the web server, end-users can grant and revoke access to areas of their own directories on the server to share information with co-workers in a controlled and dynamic way.

As a variant of the scheme, we also present a non conventional implementation of a challenge-response authentication mechanism using public-key certificates that yields the protection of a one-time password at the low cost of a CD or floppy disk, with very ordinary hardware requirements on the end-host from which the roaming user logs in. The higher security level is paid in terms of complexity on the server side. In this setting, user released certificates incorporate random information obtained from the web server and serve as one-time passwords.

We present our ideas in the framework of directory access via a web interface, and web-server based access control, mainly because this was the problem we originally addressed. The stress, though, is on the mechanism and on its features. Our ideas have wider application. The scheme could have a major role in building a simple collaborative infrastructure or it can be integrated with any authorization system for implementation of access control to services offered by an Internet portal, as support for collaborative engineering architectures.

Next section discusses our scheme in the context of related research. In Section 3 we review some concepts about certificates and certificate based access control, that we need

SAC '03 Melbourne, Florida USA

in the following. Then, in Section 4 we outline the scheme based on short-lived certificates. Section 5 describes the challenge-response variant. The implementation is discussed in Section 6. We analyze in Section 7 the security of our system. We conclude with some additional remarks and future work.

2. RELATED WORK

Secure access for a mobile user is certainly possible using specialized hardware (challenge response devices, smartcards, etc.). But we rule out this possibility since we don't want to give costly hardware to our users and we want to enable them to connect to our web-server even from a computer in an Internet Café. Authentication with certificates via a web-server would be fine, except that the certificate (along with the corresponding private key) is bound to remain in the browser's database, unless explicitly deleted by the user. But, as we said, we can't assume that our users are too careful in deploying a security protocol.

Therefore we must work with ordinary hardware and software on the client side, and still protect the user from accidental misusage of the cryptographic tools on which authentication is based.

The non-conventional use of certificates makes it possible to base authentication on X.509 certificates (and to use off-the-shelf software and no expensive hardware to do this) from any untrusted host, eliminating the risk of leaving sensible data on untrusted machines. There are access control systems based on public key authentication mechanisms and certificates, like for instance Akenti [17], [1]. Akenti provides means to implement security policies in a flexible way, but it doesn't solve our problem of a non security conscious user in an untrusted environment.

In addition, access delegation is of great interest in the domain of collaborative engineering [14]. A large amount of work is being done both in the industrial and in the academic world (see for instance [15],[4]), to provide tools for sharing information and services, and the security aspects are still under investigation [5].

The LAXCP project of the CIC Research Projects Group (the CIC is the academic consortium of twelve US Universities) has specifically addressed the issue of authentication of students on and off campus, using X.509 certificates. In their final report [3], they highlight, among other things, the usefulness of short-lived certificates. Authentication is carried out in a Kerberos based manner and certificates are used to implement authorization policies (see also further developments in [11]).

The LAXCP project is not an isolated case of use of certificates with a short validity period for authorization purposes. In [12], the authors present a framework for secure access to information servers based on temporary certificates. In their setting, users hold a secret key in a hardware token. The corresponding certificate is kept in a database on the server side. Users authenticate themselves signing a random string sent by the server; the server checks the signature using the certificate stored in the database. It then releases temporary certificates that allow for access to information servers (a public-key version of Kerberos' tickets [10]).

Short-lived certificates are also discussed in [8] for authorization in an intranet context, where user mobility is an issue. The short validity period of certificates protects the user from leaving behind sensible information.

"Personal CAs" have been introduced in [6] for authentication of components in a P.A.N., a wireless network consisting of several units located close to the user. Since the personal CA is a root CA and the trust domain is limited to the user's P.A.N., the concept only rescales the usual model of Public Key Infrastructure to a restricted environment.

3. OVERVIEW ON CERTIFICATES

We briefly review the main concepts about digital certificates that will be useful in the following.

A public key certificate is a (digital) document signed by an authority that binds a public key to a certain identity. It is also known as "digital ID".

An authority that releases certificates is called Certification Authority (CA). It consists of a (cryptographically secure) key-pair for signature and a certificate that binds its public key to its identity and states that the key-pair can be used for releasing public-key certificates. The CA certificate can be released by another CA or can be self-released. A company that needs certificates for internal use, will typically have a self-appointed CA. A self-appointed CA is also called "root CA". Any CA can release public-key certificates, but also enable somebody else to sign certificates, i.e. release a CA certificate. Finally CAs can sign Java™ applets to guarantee that they can be trusted. Signed applets can be granted privileges to write to the client's disk.

The proposed standard (and widely used) format for certificates is the X.509, described in [7]. Typical fields of X.509 certificates are: version, serial number, signature algorithm ID, issuer name, validity period, subject (user) name, subject public key information, issuer unique identifier (version 2 and 3 only), subject unique identifier (version 2 and 3 only), extensions (version 3 only), signature on the above fields. The subject's name field (`DN` which stands for "Distinguished Name") consists of a number of subfields: in the following we refer to `CN` ("Common Name") and `OU` ("Organization Unit"). Among Version 3 Extension Fields, the "Basic Constraint" extension, indicates whether the subject is a CA or not.

A hierarchy of certificates (a certificate, and the certificate of the CA that signed it, and so on up to the root CA) can be compounded in a unique file, called certificate chain. A certificate chain can be complete, to the root CA, or partial, to some intermediate CA.

A CA periodically releases a Certificate Revocation List (CRL) in which it publishes revoked certificates that have not yet expired, but that are no longer valid because, say, the corresponding secret key has been compromised, or the user has changed status, etc.

Browsers are commonly capable of managing certificates. They maintain per-user databases of personal certificates and secret keys associated to them. The secret key database is often encrypted and protected by a password, which is needed in order to modify the database and to use its contents (but not all browsers enforce the use of password protection for the secret keys).

In order to add a new user certificate and secret key pair to the browser's database, the user must request to "import" the file containing both of them. File format PKCS#12 [16] is accepted by Netscape and InternetExplorer.

On the server side, we refer to the Apache Web Server [2] with the module `mod_ssl` [13].

Apache can be configured to work at various levels of security. Configuration directives can be given at server level or even on a per-directory basis. This means that for each single directory specific access permissions can be granted. Per-directory directives are read by the server at each access. Therefore they can be reconfigured at any time and changes take effect immediately (as opposed to server level configuration directives that are read only at start-up time).

An SSL-enabled Apache web server can be configured to demand a certificate of the client that is requesting a secure (`https`) connection. In order to verify the validity of the client's certificate, it also checks the CRLs of CAs in the certificate chain in a specified directory (even CRLs of CAs that are not explicitly listed as trusted by the server). Directory access can be granted based on data in any fields of the user's certificate (and/or on general parameters of the connection).

The server looks for directory level directives in a file called (by default) `.htaccess`, in each directory of the path of the file that is being accessed, in descending order.

4. TEMPORARY CERTIFICATES

The whole functionality is based on the idea that users are granted permission to release certificates.

The Organization, has a (maybe self-appointed) certification authority; let us call it Main CA (MCA). It issues to each authorized user a child CA (which we refer to as User CA or UCA) with permission to release end-user certificates and new CAs. The UCA's private key is given to the user on a hardware token (a CD or a floppy disk) encrypted using some symmetric cipher. The secret encryption key is given separately to the user, in the form of a password. The public-key certificate of the UCA signed by MCA is also saved on the hardware token.

For privacy protection, each user and their UCAs are identified with numerical IDs, rather than their names.

4.1 Mobility

Briefly, the user Fred lets his personal CA sign a certificate with very-short (two or three minutes) time validity, and presents it to the server as he requests access.

The access authorization is given by the server after a check on the validity of the certificate and on the identity of the issuer. The `.htaccess` file might look like this:

```
SSLRequire (
    %{SSL_CLIENT_CERT_CHAIN_1} == file("UCA.pem"))
```

The directive `SSLRequire` is used to allow access to a directory based on the truth value of a boolean expression. The file `UCA.pem` contains the PEM encoded version of the certificate of Fred's UCA, and `SSL_CLIENT_CERT_CHAIN_n` is a variable that takes as its value the PEM encoded certificate of the n-th certificate up the client's certificate chain (for $n = 0$ it is the client's certificate). Therefore, the directive above instructs the server to test that the client's certificate is issued by Fred's UCA.

Notice that the check is on the certificate of the *issuer* of the temporary certificate. The validity of the short-lived certificate proves to the server that the user that is requesting access is in possession of the secret key of the UCA; the temporary certificate does not convey other information, since any UCA can sign whatever it chooses to. Moreover, checking the issuer's certificate against the file `UCA.pem` (as opposed to checking just the issuer's identity, say) serves to counter impersonation attacks in which any other user of the same company, who has a regular UCA released by the MCA, releases a CA to the name of Fred, and signs with it short-lived certificates to access Fred's area on the server.

More in detail, a secure connection is a four step process:

1. Connection to the Web Server in order to:
   (a) import into the browser the CA with which the applet has been signed (see next item), so that the applet will be trusted
   (b) download the signed applet
2. Certificate Generation, that requires
   (a) loading the UCA's private key (a PKCS#12 file) from the hardware token
   (b) getting the user's password and decrypting the UCA's secret key
   (c) compiling and signing the certificate
   (d) converting the certificate chain in PKCS#12 format
3. Certificate Importation
4. Connection

If the temporary certificates have very short time validity, then synchronization is an issue. We feel that we cannot assume that an untrusted host has accurate time. Therefore, in order to set the proper validity interval for the certificate, the applet must download date and time from the server.

We discuss in Section 6 how this can be realized and what we have done.

We anticipate that the UCA's private key is decrypted and used only by the trusted (signed) applet from the Organization, that cares not to leave sensible data behind.

4.2 Access Delegation

But to use the full potentiality of being appointed Certification Authority, Fred can also delegate to other people access to areas in his directories, maintaining the same secure authentication mechanisms and dynamically without having to inform the system administrator. The idea is simply to iterate the procedure: Fred releases a guest CA (GCA) to Mary, and Mary will use it to generate short-lived certificates following the procedure outlined above.

Let's analyze the details of how this works.

Fred releases to Mary a CA certificate to her (numerical) name `MMMMM`, with the field `OU` set to `FFFFF` (the use of the latter will be clear later), and a suitable life span.

Fred maintains in his home directory one shared directory and one private directory. (The discussion can be generalized to more than one private and one shared directories.) The `.htaccess` file in the home directory allows access to any one who has a certificate released by Fred's CA or by a GCA released by Fred's CA:

```
SSLRequire (
  %{SSL_CLIENT_CERT_CHAIN_1} == file("UCA.pem")
  OR %{SSL_CLIENT_CERT_CHAIN_2} == file("UCA.pem"))
```

Again, this counters impersonation attacks.

No one is allowed by the web server reads or writes in this directory. Basically the directory is reserved for administrative purposes.

Access to any subdirectory is conditioned to access to the home one (see [2]). Therefore only Fred or any of his guests are considered for access to the private or the shared directory.

The private directory is protected as follows:

```
SSLRequire (
   %{SSL_CLIENT_CERT_CHAIN_1} == file("../UCA.pem")
     AND %{SSL_CLIENT_S_DN_OU} != "FFFFF")
```

The first line tells the server to accept only certificates released by Fred. The second line is to check that the client's certificate is not a GCA released by Fred.

Access to the shared directory needs not be limited since the directory is shared among all users that are admitted to the home directory.

A finer sharing policy is possible, and it requires that Fred set up properly his account. Fred can share different directories with different groups of people. In order to do so, Fred must customize the `.htaccess` file in each one of the shared directories, to define who is admitted to share it. The file will look for instance like this:

```
SSLRequire ( %{SSL_CLIENT_I_DN_CN} == "MMMMM"
             OR %{SSL_CLIENT_I_DN_CN} == "JJJJJ" )
```

(i.e. the issuer must be either `MMMMM` or `JJJJJ`) in order to admit Mary (`MMMMM`) and John (`JJJJJ`).

Notice that the general admittance (i.e. to Fred's home directory) is granted based on Fred's permission (his UCA must be the last or second-last issuer in the chain), that is, the server protects Fred's data against intruders. But Fred manages the sharing of his subdirectories.

In the directory specified by the `SSLCARevocationPath` variable in the server's `httpd.conf` file, users keep the revocation lists of their CAs. When Fred's cooperation with Mary is finished, Fred revokes Mary's GCA certificate, and releases an updated revocation list. Fred uploads the revocation list to the server and the server updates the symbolic links in the directory. Mary's certificates will no longer be accepted by the server.

5. ONE-TIME CERTIFICATES

The same ideas can be used to implement a challenge-response system. The advantage over temporary certificates is that whereas an adversary can take his time working off-line in order to falsify a short-lived certificate, he can't do so in a challenge-response system.

As above, Fred connects to the access point of the web server, and fills in his own personal ID in a form. He retrieves from the server a random number computed by the server, along with the correct date and time and a signed Java™ applet. Then he proceeds as above. The only difference is that the applet writes the random number to the field `CN` of the subject of the certificate before signing it.

The server updates the `.htaccess` file in Fred's home directory to take into account the random number:

```
SSLRequire (
    %{SSL_CLIENT_CERT_CHAIN_1} == file("UCA.pem")
     AND %{SSL_CLIENT_S_DN_CN} == "RRRRR")
```

where `RRRRR` is the random number. Here the issuer's certificate must match the one saved in Fred's home directory, and the "Common Name" of the subject of the client's certificate must match the random number `RRRRR`.

In this case it is necessary to define on the server side a connection session. The end of the connection is either explicitly declared by the user, or implied by a request of access to some location with a different authentication policy, or caused by a time-out. At the end of a session the server sets the `.htaccess` file to `SSLRequire ( false )` to make sure that nobody is granted access to the directory.

Access delegation is obtained very similarly.

The `.htaccess` file in Fred's home directory is updated at each connection with a constraint that takes into account the random number:

```
SSLRequire (
  (%{SSL_CLIENT_CERT_CHAIN_1} == file("UCA.pem")
  OR %{SSL_CLIENT_CERT_CHAIN_2} == file("UCA.pem"))
            AND %{SSL_CLIENT_S_DN_CN} == "RRRRR" )
```

Access to the private and shared directories can be protected as in the previous section, except that it is no longer necessary to recognize a GCA as such, since GCA certificates don't incorporate the server's challenge.

6. THE IMPLEMENTATION

The connection procedure we detailed in Subsection 4.1 is thought as implemented using Java™ servlets and applets, for platform independency. In order to use the security and cryptography services, Java™2 [9] is necessary. For that the Java™2 plug-in has to be installed on the browser and different browsers require different but well documented configurations of the HTML page that loads the applet.

We installed an experimental web server, using Apache with `mod_ssl`. We tried various access configurations for the different settings. We wrote a signed applet to sign certificate requests and executed it, granting to it the necessary writing permissions. We successfully used our certificates for access.

In its more rudimentary form, the deployment of the logon procedure is rather cumbersome on the part of the user. Yet, most of the procedure can be automatized. The user must initiate the connection, import the certificate of the applet signing CA to the browser, grant the proper credentials to the signed applet, and provide the password that protects his private UCA key, when prompted to do so.

Specifically this implies that the applet must take care of downloading date and time, compiling and signing the certificate, importing it into the browser and proceeding to the final connection step.

Besides connection, other management services are vital to the system. They are all quite standard and should offer no real challenge. We summarize the functionalities that the system should have:

- software for preparation and signature of temporary (or challenge-response) certificates, as discussed above;
- cryptographic software for releasing GCAs, as well as support for configuration of user-defined access permissions;

- cryptographic software for revocation of GCA certificates and release of CRLs;
- software for management of CRLs, i.e for their upload to the server and for the proper update of the CRLs directory on the server;
- some user-friendly interface for general management of access delegation (including the previous three items).

Since the client is submitting a certificate chain each time it connects, it is not necessary that the server maintains a database of the UCA certificates that the MCA has released, nor of the GCAs. On the other hand, if directory sharing is supported, then CRLs of UCAs can be present on the server, maintained by the users via the software provided.

Access control can also be implemented in a server-side script (php, asp,...), which would allow for a more flexible analysis of the certificates. The Basic Constraint field, for instance would be readily available for use, which would permit a more elegant usage of certificates.

7. SECURITY

We distinguish among different types of potential adversaries, to analyze the security of the system:

Protection from a Generic Adversary.

The authentication protection is based on possession of the encrypted UCA key and knowledge of the encryption password.

As opposed to a system based on smart-cards or challenge-response password computing devices, the requirement of possession of the UCA private key is weaker than the requirement of physical possession of the hard-ware token. Indeed the contents of a floppy or a CD can be easily copied and the user might not be aware that this happened.

The UCA key, though, is also protected by a password with an arbitrarily high security level. The level of protection depends on the secret key algorithm chosen and on the length and unpredictability of the key.

The existence of end-user certificates entails the presence of end-user key pairs, but the latter are never really used. The identity proof is given by means of the UCA or GCA signature on the certificate, no matter what key pair is being certified. Therefore key-pair generation at each connection is not a hot issue.

The short life span of final certificates (or their one-time validity) protects users from replay attacks, in which certificates are seized by an adversary and their use for later (unauthorized) access is attempted.

Protection from Other UCA Owners.

Any user that has a UCA released by the same MCA is capable of releasing certificates or GCAs, to any name, generating authentication tokens that are accepted as valid by the server. The specific checks on the certificate of the issuer of the GCA or of the end-certificate (against a certificate stored in the user's home directory) protect from adversaries internal to the company.

Protection from Guests.

Malevolent guests have yet another advantage over other users, since they own a valid certificate released by their host's UCA. Indeed we had to add specific information on GCA certificates in order to distinguish them from temporary certificates issued by the same UCA. The problem is not encountered in the challenge-response version since GCA certificates can't incorporate the right random string.

Protection from Accidental Misuse.

The applet downloaded from the server automatically generates the certificate. Only a temporary certificate is imported into the browser. The certificate doesn't contain relevant data and after its expiry is useless.

The user is only requested not to keep password and hardware token together, and not to entrust them to anyone, and to inform promptly the MCA if the password is compromised and/or the hardware token is lost or stolen. This is the minimum requirement for any authentication system based on "something the user knows" and "something the user owns".

All users who regularly carry out certificate operations using the software provided by the company don't have to worry about clean-up procedures, or extra security measures.

The possibility is still left open that alternative software is made available (and indeed the skilled user has a wide choice of cryptographic tools) for signing certificates with longer validity, or guest CAs that don't have the `CN_DN_OU` field properly set, etc. This kind of misuse can be prevented only to a certain extent, but we feel that it can be considered equivalent to lending the hardware token or disclosing the password to unauthorized users.

8. CONCLUSIONS AND FUTURE WORK

In all certificate related applications we have seen in literature, user and CA domains are clearly distinct. Our scheme proposes an unconventional way to look at certificates. An analogy could be a debit-card system that enables debit-card owners to release their own short-lived or one-time debit-cards (tokens). Users would carry only the tokens in their wallets and wouldn't need to worry much about forgetting the token in the cash-machine or about losing it. The example can be carried over to delegation too.

The natural evolution of our research will bring us to an in-depth analysis of the potentiality of our system as a basis for authentication in collaborative infrastructures.

Besides, we intend to complete the current project with the implementation of all the services mentioned in Section 6, automatizing the procedure as much as possible, for user-friendliness.

At the same time, we must address some pending issues. For instance, expired temporary certificates are bound to pile up in browsers unless users remember to delete them every now and then. In our hypotheses, we can't depend on users for such garbage collection tasks.

On machines from which the server is accessed often, one might think of simplifying the procedure by implementing the client-side software as a browser plug-in. Although the plug-in solution limits the mobility, it can be useful as an alternative approach for users or for moments in which mobility is not an issue.

Also, we don't like the non-conventional use of the field `OU` in the Distinguished Name of the certificate's subject, especially since the information we need is already contained in

the X.509v3 extension field Basic Constraints. To the best of our knowledge, there is no environment variable in Apache with mod_ssl whose value is that field (we also posed the question to the mailing list `modssl-users@modssl.org`, but to no avail). It should be possible to add an environment variable, if the one we need doesn't exist, since the information is clearly available. The other solution is to process certificate information with a server-side script that controls access, as suggested at the end of Section 6.

Acknowledgements

We would like to thank Giovanni Porcelli for having brought the original problem to our attention and for many useful discussions.

9. REFERENCES

[1] Akenti: Distributed Access Control, `http://www-itg.lbl.gov/security/Akenti`

[2] The Apache Software Foundation, `http://www.apache.org`

[3] Committee on Institutional Cooperation, Research Projects Group: Local Authentication with X.509 Certificates Project (LAXCP). `http://www-snap.it-services.nwu.edu/CICRPG` (1999)

[4] M.R. Cutkosky, J.M. Tenenbaum, J. Glicksman. Madefast: an Exercise in Collaborative Engineering over the Internet. Communications of the ACM (to appear) `http://madefastr._stanford.edu/ACM_paper.html`

[5] C. Eliopoulis. Balancing the Requirements: Collaborative Security. The Edge Perspectives, 1, `http://www.mitre.org/pubs/edge_perspectives`, 2000.

[6] C. Gehrmann, K. Nyberg, C.J. Mitchell. The personal CA–PKI for Personal Area Network, *IST Mobile & Wireless Telecommunications Summit 2002*, `http://newton.ee.auth.gr/summit2002`, 2002.

[7] R. Housley, W. Polk, W. Ford, D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) profile. *RFC 3280*, 2002.

[8] Y-K. Hsu, S.P. Seymour. An Intranet Security Framework Based on Short-Lived Certificates. *IEEE Internet Computing*, 2: 73–79, 1998.

[9] Java™2 Software. `http://java.sun.com/java2`

[10] Kerberos: the Network Authentication Protocol. `http://web.mit.edu/kerberos/www`

[11] O. Kornjevskaia, P. Honeyman, B. Doster, K. Coffman. Kerberized Credential Translation: A Solution to Web Access Control. *Proceedings of the 10th USENIX Security Symposium*, 2001.

[12] López,D., Reina, M.: Providing Secure Mobile Access to Information Servers with Temporary Certificates. *Computer Networks*, 31:2287–2292, 1999.

[13] Mod_SSL, The Apache Interface to OpenSSL. `http://www.modssl.org`

[14] L. Monplaisir, N. Singh (eds.). *Collaborative Engineering for Product Design and Development.* American Scientific Publishers, 2002.

[15] A. Pawlak. Collaborative Engineering – A New, Emerging Paradigm of Engineering Work Based on Internet. `http://www.man.poznan.pl/ist/isthmus/programme`

[16] RSA Security: Public-Key Cryptography Standards. `http://www.rsasecurity.com/rsalabs/pkcs`

[17] M. Thompson,W. Johnson, S. Mudumbai, G. Hoo, K. Jackson, A. Essiari. Certificate Based Access Control for Widely Distributed Resources. *Proceedings of the 8th USENIX Security Symposium*, 1999.