SIMS: A Secure Information Management System for Large-Scale Dynamic Coalitions[1]

Keyu Jiang and Partha Dasgupta
Arizona State University
keyu.jiang@asu.edu, partha@asu.edu

Abstract

When two (or more) entities (or members) enter into a coalition, they agree to share information, resources and other assets according to some set of negotiated rules. This paper addresses the issue of controlled and secure information sharing.

Each member may have a large number of agents (people) who run programs that access information from the large number of servers run by the other member. The problem arises in managing the authentication and the access control at these service points. The issues are technical, as well as administrative. Compounding the problem is the large number of autonomous information servers that contain the information published by a single member. Administering and securing these is in reality intractable.

We present a solution to the secure information-sharing problem, by separating the authentication function from the data access function. Then, by having only one authenticator per member and the use of digital certificates we show how a multiplicity of information sources can be managed and secured.

1. Introduction

A coalition is a collection of entities that collaborate for some major purpose or goal by cooperating, exchanging information sharing resources and pooling capabilities. Resources and capabilities may include information, assets, computing infrastructures, military capabilities and so on. Sharing, collaboration and exchanges have to be performed in a secure and controlled fashion. A Dynamic Coalition is a type of coalition, in which the membership of the coalition and trust relationship between members could dynamically change, and the non-symmetric trust relationships may exist.

This paper focuses on the issue of sharing information among the participants of a dynamic coalition. This information is typically in the form of files, possibly published via a web-style interface. However there are rules (which must be strictly and infallibly enforced) that control who (and under what circumstances) can access the information (or update the information). The management of the access control, as we shall show later, can be a daunting challenge. It is also of concern that the access management system may be compromised using unknown loopholes and/or bugs.

In the following sections, we will first define, in brief, our view of and terminology for a dynamic coalition. Then in section 3 will present the problems and possible shortcomings of secure information management and motivate the need for an innovative solution. Section 4 presents a new approach that separates the authentication and access functions and thus provides a significantly more manageable and secure solution. Section 5 presents an architectural design of the implementation of the proposed system and section 6 presents results from the actual implementation of the prototype. Section 7 discusses enhancements that are in progress and approaches to reduce vulnerabilities.

2. Dynamic Coalitions

A dynamic coalition, as mentioned above, is a collection of "members", where the members consist of organizations, companies, governments or such large entities. To share resources, the members in a coalition need to have trust relationships between each other. According to some *a-priori* (but dynamic) agreements between each pair of members, they define their own resource access policies. In addition to members there are a significant number of other entities participating in the coalition, and these include people, computers, information, programs, processes, rules and so on. We define the following terminology to refer to these entities in a consistent manner [Da*99]

- ***Member:*** A member is a self-controlled and independent entity in a Dynamic Coalition. A member decides to join or leave the Dynamic Coalition autonomously. A member may also be

[1] This research is partially supported by grants from DARPA (F30602-99-1-0517, N66001-00-1-8920), NSF (CCR-9988204) and Microsoft, and is a joint research effort between Arizona State University and New York University.

disenfranchised by other members. *Example: a country in the United Nations.*

- ***Agent:*** An agent is an identifiable and an accountable actor. An agent belongs to a specific member and each member may possess potentially unlimited number of agents. In general, and agent is a human being or "intelligent" computerized device with some autonomy. *Example: an interactive human user*.
- ***Delegate:*** An activity (mostly computerized) of limited lifetime (at most the lifetime of a coalition) created by an agent, or by another delegate. An agent can create any number of delegates and then these delegates may create more delegates forming a forest, each tree rooted at a delegate created by an agent. The agent who created the root delegate, owns all the delegates in the tree. *Example: a process*.
- ***Resource:*** Any entity accessible to agents and delegates. *Examples: File, Database*.
- ***Asset:*** A physical resource or host for delegates and objects. The asset is owned by a particular member. *Example: A computer.*
- ***Directive:*** The partners, agents, and delegates operate, adhering to a set of directives. The directives are access control rules which specify what resources can be accessed, and which trust levels have to be obeyed under what circumstances

While Agents make the high level decisions on how activity is managed, delegates do all actions inside a coalition. The delegates are actively running computations. These delegates execute on assets and access objects. Suppose a delegate *D* is executing on an asset *X*. An agent *A* owns the delegate *D and* A is owned by the member M. The owner of the asset *X*, however, need not be M. When the delegate *D* accesses a resource *R* (on *X)* the directives that specify access control between M and the owner of *X* determine if and how *D* can access *R*. Thus the directives contain rules for every possible access pair of delegate and resource.

Of course, creating and storing directives for every delegate-resource pair is obviously infeasible. We have devised practically feasible methods of defining these access control rules [Ji00]– a description of these techniques are beyond the scope of this paper. Hence, in this paper we assume, that these rules are maintained and are publicly available.

The thrust of our research in the Dynamic Coalitions area is to develop and prototype management systems that handle the infrastructure of Dynamic Coalitions. That is, we are inventing mechanisms, policies, methodologies and techniques that will enable the actual control and execution of a real dynamic coalition. Our research is expected to provide a set of software modules that can be used to build the infrastructure needed for cooperating members along with their agents, delegates, objects and assets. In this paper we focus on *one* aspect of the Dynamic Coalition infrastructure, that is the management of shared information. In order to do this, we will rely on some well-known underlying mechanisms such as encryption (public and private key), digital certificates, signatures and one-way hashes.

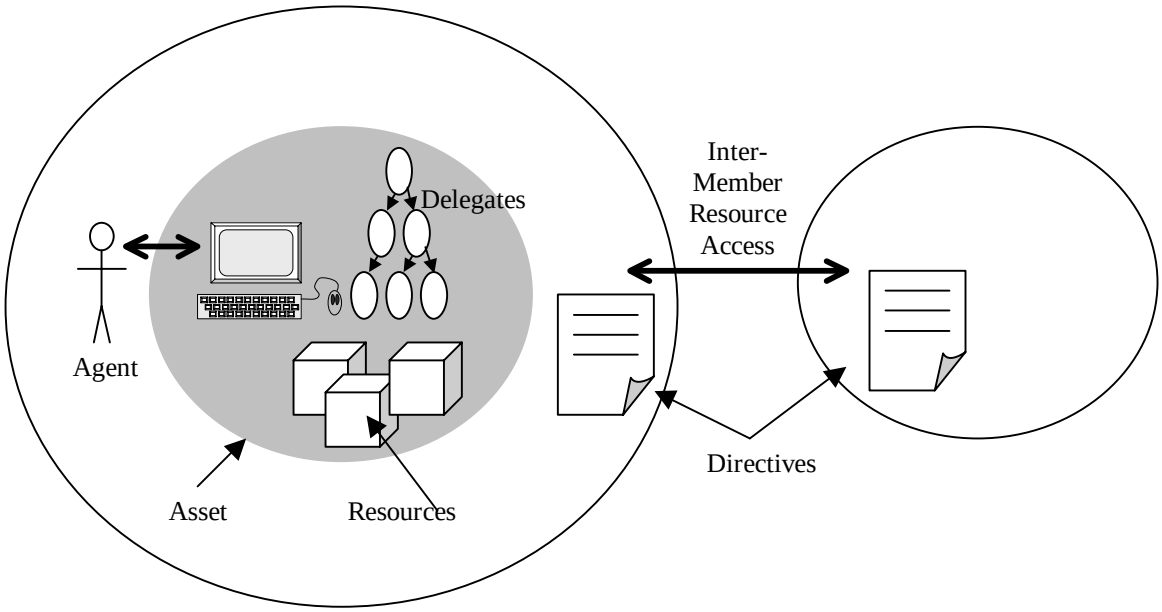

Figure 1: The elements of the Dynamic Coalition Management System

3. Access Control for Information

We now present the secure information sharing problem. To illustrate this, let us start with an example. Suppose there are three corporations – *AirMan* manufactures aircraft, *PartMan* provides aircraft parts to *AirMan* and *ModelShop* provides CAD support to both *AirMan* and *PartMan*. These three corporations form a coalition and decide to share information according to a comprehensive set of rules, a few of which are stated below for illustrative purposes:

- All Engineers working for AirMan should have access to all part specifications and prices of PartMan parts.
- PartMan engineers can also access AirMan requirement databases.
- Management personnel at AirMan can access summary personnel files of PartMan, but not the parts information.
- Various design data from ModelShop are available to AirMan and to PartMan, however some data is visible only to AirMan and some to PartMan (and some to both).

Given the above scenario, we wish to provide a secure, yet simple to use and maintain, information management system to enable such collaborations.

First, we choose to make the data available on file servers (could also be web servers). The people who are authorized to see the data can then access the data over the Internet (securely). However, this produces an instant complex problem. The issues are:

1. If James, an engineer at AirMan wants to access some parts information at PartMan, how does the PartMan file server authenticate James?
2. If James gets promoted to manager, he looses his access rights to Parts, but gets access rights to some Personnel files. How do the PartMan file servers know of this change?
3. All file servers (and there could be thousands of them) have to be intimately aware of all personnel and personnel changes at all members of the coalition – this is infeasible.
4. James needs to have accounts at all file servers of PartMan and hence probably has hundred of different id's and passwords (having the same id and password is a security risk). This is impractical.
5. It is not a feasible task to ensure the security of thousands of file servers on the Internet from attackers – all of the file servers cannot be adequately secured.

Any solution for this problem based on conventional authentication and access control methods are doomed to be complex and hence prone to failure. In fact the scale and dynamic nature of the coalitions; and the dynamic nature of the rules and the dynamic nature of personnel classifications and such other dynamic properties makes the problem quite intractable.

Our solution however is interestingly simple. By disassociating the authentication function with the data dissemination function, we can take the access control responsibility off of the file servers and thus drastically reduce the complexity of access control. Preliminary results of this idea are reported in [Da*99]

4. Separating Authentication and Access

As stated earlier our solution consists of the separation of authentication and data access, and this approach is motivated by the premise that the security of information (or files) cannot be entrusted to file servers.

In our design, file servers are "dumb" entities that

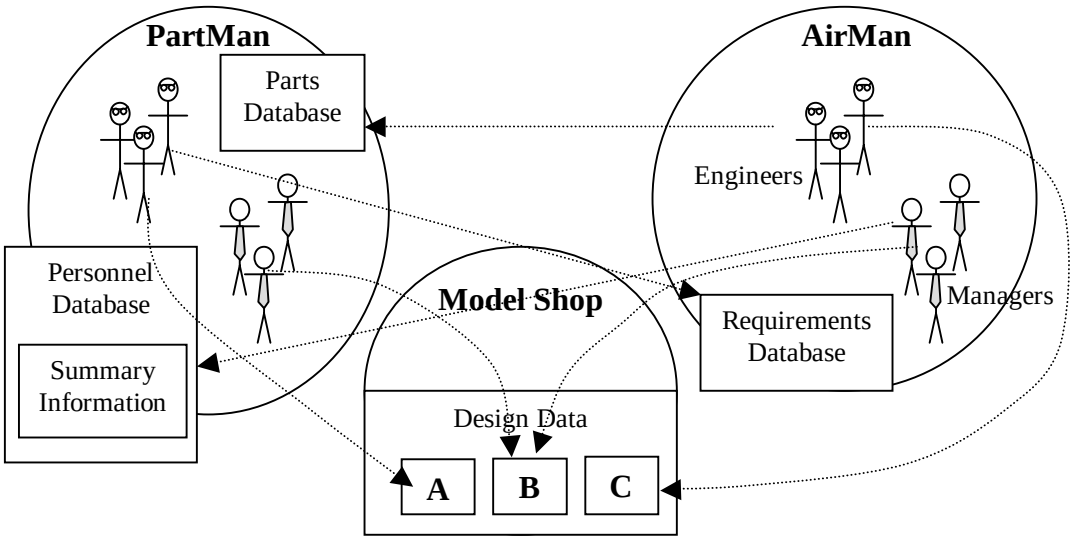

Figure 2: Access provisions for the example in Section 3

store files and deliver the files upon request. In order to ensure the security of such files, all files that are stored on file servers are encrypted with strong keys. There is one unique key per file. The keys are *not* stored on file servers.

Using this approach, a file is no longer useful without the key. Hence the file servers need not authenticate the service request. If a delegate requests a file, the file server delivers the file without regard to which agent owns the delegate, or where the delegate is running.

In order for the delegate to use the information in the file, the file needs to be decrypted, and hence the key has to be acquired by the delegate. The resulting security of the information system resides in the mechanisms used for key distribution. The key distribution is provided by having a repository of all the keys for all the files belonging to a member. We call this repository the *key server*.

Actually the authentication problem that we separated out of the file servers is now the job of the key server—and all the same problems seem to exist. However, we argue that this is a better solution (than having the file servers authenticate delegates) for the following reasons:

- While performance and other issues necessitate the use of many file servers, all the key handling can be centralized into one (possible two—or a small number) of key servers.
- Managing the security of one (or a small number) of key servers is much more tractable that ensuring the secure operation of possibly hundreds of file servers
- When the trust relationships change, the directives have to be updates only at a few key servers and not at all the file servers.
- Maintaining audit logs and monitoring intrusions and other compromise are possible when the number of secure sites is few.

In order to further simplify the management of authentication, we make use of digital certificates. Our scheme of digital certificate based authentication allows us to quickly authenticate a delegate and determine its access privilege for a particular object it wants to access.

4.1 Digital Certificates

As is well known, a digital certificate is a trusted proof of identity. A well known, trusted authority called the Certificate Authority or CA issues a digital certificate. The CA grants the digital certificate to an agent (or entity) after verifying the existence and identity of the agent, often through some out of band means. The certificate itself is a set of data items, such as the identity of the agent, class of the agent (e.g. engineer, manager, student) level of trustworthiness and so on; and the public key of this agent. The certifying authority then signs this data, using its private key. The private key corresponding to the public key in the certificate is kept secret by the owner of the certificate (generally the agent).

In our design, every coalition has a designated, trusted Certificate Authority, called the Coalition-CA. The Coalition-CA issues certificates to subordinate CAs administered by members, thus identifying these CAs. These other CAs are CAs run and maintained by each member of the coalition and are called member-CAs. The public key of the Coalition-CA is well known and publicly available and verifiable (maybe through out of band channels). The public keys of the member-CAs are known by accessing the certificate of that CA. The level of trust a member A has for the member B depends upon what A knows about B's procedures for keeping its CA secure.

In addition to one member-CA, a member may have subordinate CA's, however certificates issued by the subordinate CA's may not enjoy the same level of trustworthiness as the member-CA. Again, the level of trust for such situations are contained in the directives. For the scope of this paper, we will assume there is only *one* member-CA for each member.

The member-CA issues certificates to all agents and services of the member. The agent certificate contains not only the agent's identification, but also its affiliation to the member, its class, and its capabilities

4.2 Key Server and Access Control

Recall that all sharable information is placed on some file server, in an encrypted form. No access control is imposed on the file and anyone (of course, even unauthorized agents) can obtain a copy of this file. It is assumed very strong encryption is used and thus obtaining the encrypted file is of no use to any agent.

Now we discuss, in concept, how an agent, authorized to obtain the information in the file, gets the key to decrypt the file. Suppose a delegate D belonging to an agent A needs information contained in file F. F is stored at a file server S (in encrypted form), in the domain of member M. The steps are:

- The delegate D downloads a copy of file F from the server S. (Steps 1 and 2 in Figure 3.)
- The delegate D obtains a copy of the certificate of agent A and sends it to the Key Server of the member M. (Step 3 of Figure 3.)
- They key server of M then authenticates D by using a challenge response algorithm by which D proves it has access to the private key of A, and hence is authorized by A to act in his/her behalf.
- The Key Server now uses the identification and classification data in A's certificate and consults the rules in the directives to decide whether D should access F or not.

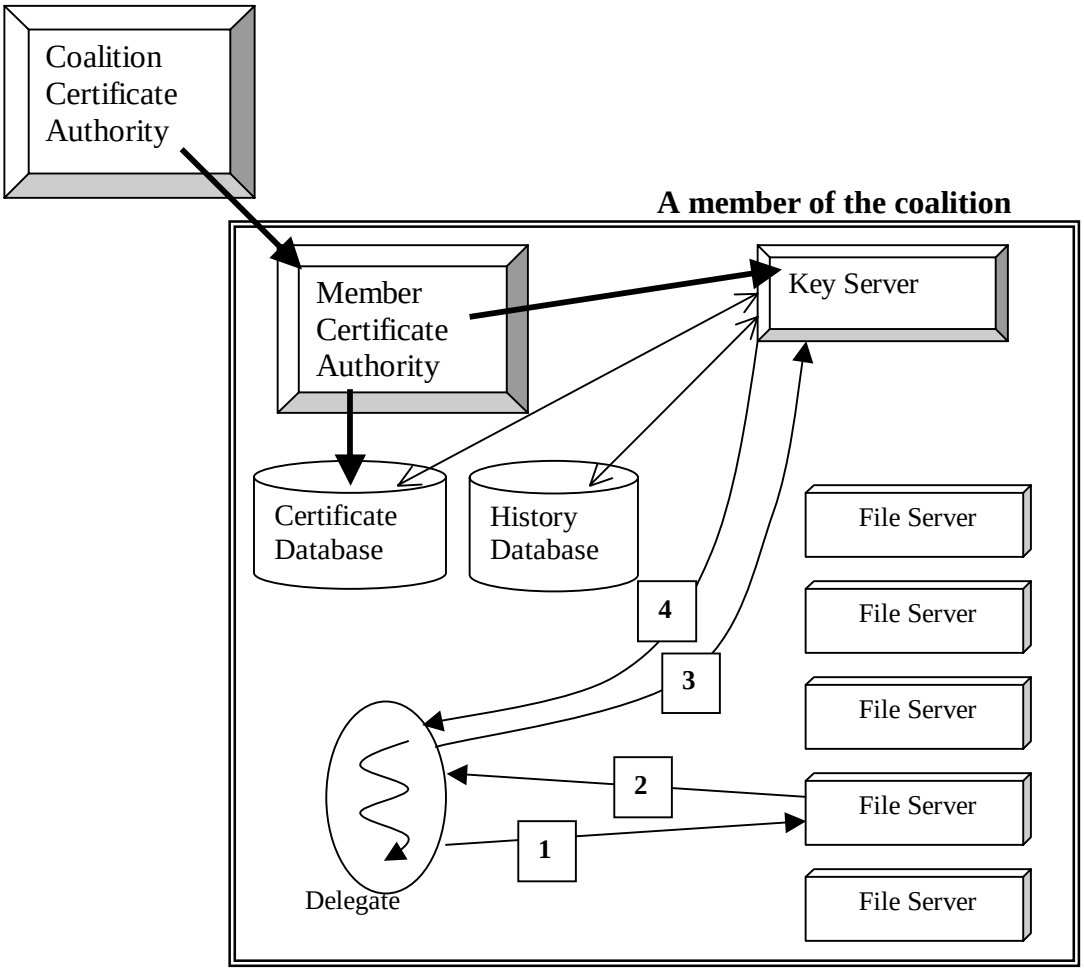

Figure 3: The functional modules of the Dynamic Coalition Management System

- If access of F by D is permitted, the Key Server sends the key of F to D (of course using a secure communication channel). (Step 4 of Figure 3.)

As is obvious, in the above scheme there is no authentication at the file server and there is no communication between the key server and the certificate authorities. Also, due to the certificate scheme, the key server of M does not have to be cognizant of the personnel changes of any other member in the coalition—the key server can determine the standing of an agent and its access rights form the certificate and the directives. Note that it is impossible for an agent to tamper with the information in its certificate (unless if the CA's private key is compromised). Also a stolen certificate cannot be used, as it will fail the challenge-response test, which needs the private key of the agent owning the certificate.

We also assume that the integrity and confidentiality of messages between servers are maintained using conventional encryption, signatures and MAC techniques.

5. Architecture of SIMS

So far we have presented a conceptual design of a secure information management system that uses encrypted files to store information and controls the information access by controlling the key distribution. We now describe the architecture, the protocols and the implementation of *SIMS*, the *S*ecure *I*nformation *M*anagement *S*ystem. In order to describe SIMS, we first present some details of the architecture of a dynamic coalition system and then show how SIMS fits into this environment.

The Dynamic Coalition Management System we are building consists of one Coalition-CA (as described before) and a set of modules on a per-member basis. Each member maintains servers for the following (see Figure 3):

1. *A Member CA*.
2. *A History Database*. This database contains records of "aberrant" behaviors – that is a record of denied access and other faults on a per-agent basis. The information in this database is used as a "credit-report" when judging trust relationships.

3. *A Certificate Database*. This contains all the certificates issued by the member-CA. This allows any server (or service provider) to get a certificate of any agent requesting service.
4. *A Key Server*: The server responsible for authenticating requests and validating the access privileges of the agent.
5. *A set of file servers*: These provide access to the shared information.

The above set of modules form the basic infrastructure of the Dynamic Coalition Management System and is closely relevant to the secure information management system we are describing in this paper. SIMS is heavily dependant on the certificate services, file services and the key services. It also depends upon the Certificate Database and the History database. Other parts of the system include directive databases, intrusion detection systems, trust level negotiators and so on, which are outside the scope of this paper.

SIMS, as described in the following subsections, is a flat file system, which stores files along with file names on a set of servers. It is the responsibility of the reader to know the name and location of each file it wants to read. Also, filenames are not hidden. The reason for such an exposition is to eliminate the complexities of directory services, name hiding and such artifacts from the crux of the design. A brief discussion of adding features to the basic design is in section 7.

The basic concepts and working of SIMS has already been discussed, in the next few subsections we show the steps needed for creating, updating and deleting files in SIMS.

5.1 Creating a file

In order to publish some information, an agent has to create a file, containing this information. A delegate running on behalf of the agent does the actual creation. We assume, that if delegate D running on behalf of Agent A does the file creation, and if A belongs to member M, then the file is created and stored on assets and servers belonging to M. This assumption can be relaxed, but makes the discussion more complex. Note that the creation of a file is not a security risk – any agent belonging to a member can create a file and publish information it wants to publish. The steps involved are:

1. The delegate D produces the file F and then encrypts it using a key K_f producing *e*F.
2. D then creates an *access control list* (ACL) for the file. If this ACL is null, the coalition directives apply for access control, else the access control list supersedes coalition rules.
3. D sends the filename, size, ACL in a secure message to the file server FS.
4. FS sends D a port number FS and expects the file to be received at this port.
5. D sends *e*F to FS. MACs are used to ensure uncorrupted transmission. *This communication does not have to be encrypted as the file is already encrypted.*
6. D sends the identity of the agent A and the filename, ACL and K_f using secure communications to KS.
7. KS retrieves the certificate for A from the certificate database, and then authenticates D.
8. If D passes authentication and is in good standing (according to the history database) then KS stores the filename, ACL and K_f in its data store.

The above procedure allows a delegate to store a file on a file server of its own member. The encryption step (as we will show later) is one of the expensive steps, but note that this step is performed on the machine that the delegate is running upon and does not cause loads on the file server or network. The file transmission is also time consuming, but this is the normal function of the file server – scalability can be achieved by increasing the number of file servers. The transactions with the key servers happen only once per file creation and is of low overhead.

5.2 Reading a file

Reading a file from a file server is expected to be the most common operation in the system. This is also the simplest operation in SIMS. The steps are:

- The delegate D that wishes to read a file F, sends a request with the filename to the file server FS.
- FS sends the entire encrypted file *e*F to D. *This communication does not have to be encrypted as the file is already encrypted.* Also, MACs are not necessary as the delegate can discover tampering, see section 7.
- D sends a key request to the Key Server, KS with the identity of the agent A and file name.
- KS retrieves the certificate of A and then checks the directives and the ACL associated with F to ensure D can have the key K_f.
- If the directive test passes, KS authenticates D.
- If authentication passes, KS sends K_f to D (securely).
- If any of the above two tests fail, a record of the failure is entered into the history database.

As before, the performance of the read operation is determined by the download and decryption times. Decryption is a local operation, and the download is a raw download – does not involve encryption. Hence the performance is quite close to the performance of unsecured data access.

5.3 Updating a file

The SIMS is expected to be used in a read-mainly situation and is designed with this in mind. The read operation is the most efficient and straightforward. The update operation is the most complex. There are two approaches to manage updates.

The first approach is to use immutable files, i.e. a file once created cannot be updated. File updates are done by actually creating a new version. Since creation is supported, the updating delegate attaches a version number to the updated file, essentially creating a new file. It is the responsibility of the reader to determine which version to read and which version to trust.

The immutable file strategy is cumbersome and user-unfriendly. However it is the most secure, as it puts no intelligence into the file server. The second approach allows files to be actually file updated, but this needs the file server to have a securely stored private key. Compromise of this private key can cause a limited (but detectable) security risk.

Suppose a delegate D needs to update a file F stored at file server FS

- D creates an updated version of F and then selects a new encryption key K_f and encrypts the file producing *e*F
- D sends an "update request" to the Key Server KS along with the identity of the owning agent A the filename and the key K_f.
- KS checks the certificate of A, authenticates D and then checks the directives and ACL of F to ensure D is allowed to update F.
- KS stores the new key and informs the file server FS of the update and obtains an "*update port number*", securely.
- KS informs D of the *update port number*
- D sends *e*F to the update port number and FS replaces the old file with the new one.

The possible weakness in this scheme is the need for a file server public key. In order for the key server to communicate securely with the file server, the file server must have a public-private key pair (i.e. must have a certificate). If a hacker steals the file server certificate then the security of the file server is compromised and files may be corrupted. This vulnerability does not exist in the other cases:

1. In the file creation step there is also a need for secure communication with the file server, however if the private key of the file server is compromised, no existing file can be changed – only new files may be added which are spurious.
2. In the immutable file scheme, a new but spurious, version of a file can be added, but no existing version can be deleted.

In any case, please note that a file server can be compromised and all its files deleted (or changed) but this can be detected and files restored from backup. No information can be *stolen* by compromising a file server.

6. Prototype and Performance

We have implemented SIMS as a prototype with some limitations. Firstly, the directive handling scheme has not been implemented; hence the ACL scheme is used as the sole enforcer of access rights. Secondly, several planned enhancements are not completed and are described in section 7.

The implementation of SIMS runs on Windows 2000. It uses the Windows Crypto API for generation of encryptions, hashes, signatures and so on. It uses the Win-2000 certificate server for generating certificates for agents and servers. The implementation consists of all the servers described above, that is file server, key server, history database and certificate database. The servers are Windows console applications and a set of library routines exists for handling file creation, reading and updates. The operation of the system has been tested and it works as expected.

The performance of the system is also as expected. Encrypting or decrypting a 1 MB file on an 800Mhz system takes 0.27 secs (a 10MB file takes 1.7 secs). Downloading from a file server is the slowest operation, at 1.5 secs for a 1MB file (3.4 secs for 10MB). Comparatively, the key server request is fast. A request for a key, including database lookup and authentication takes 0.02 secs. Note that these numbers are for a preliminary, un-optimized implementation and are expected to improve substantially if the software is optimized.

7. Enhancements and Vulnerabilities

In the following sections we discuss some observed shortcomings or perceived vulnerabilities and show how to fix them, if necessary.

File Names: The main shortcoming of the above design is the use of a flat file system with publicly accessible names, that is, all file names are visible to everyone. The solution to this problem is to publish the directory information about files in the SIMS, using SIMS itself. This will ensure, only those that are authorized to see a certain file name, can see that file name. The idea is simple—all files stored on the server are assigned random names. These system-generated names are stored in a hierarchical directory system where the actual user defined names are present. The directory itself is a set of files, one file per directory. In order to traverse the directory, the delegate has to fetch each directory file and decrypt its contents. This way, each directory can have directives or ACLs specifying the access rights.

File Server Compromises: A vulnerability of the file server scheme is what happens if a particular file server is compromised. The compromised file server can be then made to give out false or misleading information. Building in a detection mechanism as follows can stop such errant behavior. We add to each file the following:

1. A hash of the file contents that is signed by the creator.
2. A description of the file, and its contents, signed by the creator.

Then a reader can securely verify the received contents and detect any tampering. Note that the signatures and hashes are added before the file is encrypted.

Performance: The performance of the system is somewhat limited by using the whole file approach, that is the files are transmitted in totality – unlike conventional file service where files are fetched in blocks, on demand. We are working on a block structured file system where each block can be requested on demand from the file server. This is however different from normal file service as each block must be verified and decrypted independently—hence the hashes, signatures and descriptions have to be attached separately to each block. Note that partial updates of blocks have to be handled using different update mechanisms than what we have discussed.

Access Right Overriding: Suppose Alice creates a file and Bob is not allowed to access it. However, Alice decides to provide Bob with temporary access. It should be possible for Alice to provide Bob with a signed certificate that the key server will honor for a limited lifetime. This functionality is being added.

Denial of Service: In an open "trust-based" network like TCP-IP, denial of service attacks are very difficult if not impossible to prevent. However we can minimize the effects of such attacks. Simply flooding the network with packets is an attack we cannot prevent or insulate against. In our system, however unauthorized file downloading is made difficult by the directory structure. The attacker has to know the name of the file – which can only be obtained by downloading and decrypting the directory and hence attackers not within the member organizations cannot launch this attack.

Sending spurious requests can launch a denial of service attack against the key server. Our design discourages such attacks—because of the need for authentication. All requests to the Key Server needs a sender identifier. Wrong or forged identifiers are easily detected and a high traffic of these immediately causes the attack to be detected.

Enhanced ACL: The ACL used for access control can be enhanced for synchronization purposes. For example, a delegate, after authentication with the key server, can request notification of modification if a certain file. The key server enters this in the ACL of the file. Next time the file is updated, the key server can detect the notification request in the ACL and notify the requestor. Other such enhancements are possible and are being researched.

8. Related Work

The majority of the work in information security is about authentication techniques and data security using ACL's and authentication. Separation of authentication and access is a new concept; the only other work that uses this idea is the secure file system or SFS [Ma*99, Ma00, Fu*00]. In SFS the main focus is to ensure that a file server does not spoof the client. Hence SFS used Self-certifying pathnames and this ensures that the client can easily certify the server. In our case however, the problem is reverse, we need the servers to ensure that information does not fall in the wrong hands.

Other related research includes encrypting file systems such as PGP and Window 2000 encrypted file systems [Ho99]. These systems encrypt and decrypt storage on the fly such that the data on the disk is always encrypted. Our system also keeps the disk encrypted as a side effect.

Work in Dynamic Coalitions is a new area. Some results use secure group communications [Ca*98] and are orthogonal to our research.

9. References

[Ab*93] M.D. Abrams, Renewed Understanding of Access Control Policies, *Proceeding of the 16th National Computer Security Conference*, pp.87-96, Oct. 1993.

[Ca*98] Caronni G, Waldvogel K, Sun D., and Plattner B, Efficient Security for large and dynamic multicast groups, *Proceedings Seventh IEEE International Workshop on Enabling Technologies: Infrastucture for Collaborative Enterprises*, 1998.

[Da*99] P. Dasgupta, V. Karamcheti, and Z. Kedem. Efficient and Secure Information Sharing in Distributed, Collaborative Environments. In *Proc. of 3rd Intl. Workshop on Communication-based Systems*, April 2000.

[Fu*00] Kevin Fu, Frans Kaashoek, and David Mazières, Fast and secure distributed read-only file system, *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, 2000.

[Go*96] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A secure environment for untrusted helper applications, *Proceeding of the 16th Usenix Security Symposium*, July 1996.

[Gr*99] R. Grimm and B. N. Bershad, Providing Policy-neutral And Transparent Access Control in Extensible Systems, June, 1999.

[Ha*95] Martin Hardwich, Blair R. Downie, Mike Kutcher, and David L. Spooner, Concurrent Engineering with Delta Files, *IEEE Computer Graphics and Applications*, PP.62-68, Jan. 1995.

[Ho99] Jerry Honeycutt, The Encrypting File System, *Introducing Microsoft Windows 2000 Professional*, pp.230-235, 1999.

[Ke*98] S. Kent, and R. Atkinson, Security Architecture for the Internet Protocol, RFC 2401, Internet Engineering Task Force, Nov. 1998, ftp://ftp.isi.edu/in-notes/rfc2401.txt.

[Ji00] Keyu Jiang, Dynamic Coalition Management Systems, Ph.D. proposal, Arizona State University, December 2000.

[Ma00] David Mazières, Self-certifying file system. PhD thesis, MIT, May 2000.

[Ma*98] D. Maughan, M. Schertler, M. Schneider, and J. Turner, Internet Security Association and Key Management Protocol (ISAKMP), RFC 2408, Internet Engineering Task Force, Nov. 1998

[Ma*99] David Mazieres, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel, Separating Key Management From File System Security, 17th ACM Symposium on Operating Systems Principles (SOSP '99) PP.124-139, Dec. 1999.

[Rj*98] R. J. Hayton, J. M. Bacon, and K. Moody, Access Control In An Open Distributed Environment, *1998 IEEE Symposium on Security and Privacy*, pp.3-14, 1998.

[Sc96] Bruce Schneier, Applied Cryptography, second edition 1996

[Sp*99] Ray Spencer, Stephen Smalley, Peter Loscocco, Mike Hibler, Dave Andersen, and Jay Lepreau, The Flask Security Architecture: System Support For Diverse Security Policies, Proceedings of the 8th USENIX Security Symposium, August, 1999.

Sponsor Acknowledgment: Effort sponsored by the Defense Advanced Research Projects Agency, under agreement numbers F30602-99-1-0517 and N66001-00-1-8920. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon.

Sponsor Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.