# Key management for encrypted data storage in distributed systems[*]

Ludwig Seitz Jean-Marc Pierson
Lionel Brunie
LIRIS, CNRS FRC 2672, INSA de Lyon
7, av. Jean Capelle, 69621 Villeurbanne cedex, FRANCE
{ludwig.seitz, jean-marc.pierson, lionel.brunie}@liris.cnrs.fr

## Abstract

*Confidential data stored on mass storage devices is at risk to be disclosed to persons getting physical or administrator access to the device. Encrypting the data reduces this risk, at the cost of more cumbersome administration. In this publication, we examine the problem of encrypted data storage in a grid computing environment, where storage capacity and data is shared across organizational boundaries. We propose an architecture that allows users to store and share encrypted data in this environment. Access to decryption keys is granted based on the grids data access permissions. The system is therefore usable as an additional security feature together with a classical access control mechanism. Data owners can choose different tradeoffs of security versus efficiency. Storage servers need not to be trusted and common access control models are supported.*

***Keywords:*** *Secure storage, key management, access control, distributed storage, grid storage*

## 1. Introduction

A large variety of sophisticated protocols for secure transfer of data over insecure networks exists today (e.g. SSH[15], TLS[4], IPSec[8]). However most successful attacks that result in disclosure of confidential data don't occur while data is transmitted over a network. In fact data on a mass storage device is a much easier target. Getting access to such devices through social engineering, weak passwords or security holes in the access control system is a relatively simple task compared to successfully intercepting and interpreting a network communication.

If we go one step further and enter the world of distributed or grid computing, where data may be stored outside the administrative domain of its owner, the problems of secure storage and access control become even more obvious.

Grid access control mechanisms, as presented for example in [13] and [1] only protect the data as long as the attacker does not get physical or administrator access to the storage system. If an attacker gets such access, all data is freely accessible to him. However in some applications, for example in medical data grids, protection against this risk is required in addition to the usual access control mechanisms, because of the sensitive nature of the data.

Bulk data encryption provides means of reducing the above risk. Several programs exist that provide such services for single file encryption (e.g. PGP[16]) or for encrypted file systems (see section 2). However key management has always been a problem for sharing encrypted files. The main reason for this is the longer lifetime required by encryption keys for storage in contrast to keys for communication. They need to be securely stored and authorized users must have a way to access them. A distributed environment like grid computing, where users from different domains share storage space and data, adds a new level of complexity to the problem. Indeed users may have to grant access to confidential data to some other users not necessarily of the same administrative domain. This raises the question how authorized users will gain access to decryption keys for encrypted data and who will be responsible of providing those keys. Therefore we need a key management scheme, adapted to the grid environment. Authorized users must be able to access decryption keys, even if the data owner is from a different administrative domain. Data owners must not be required to trust the data storage server, since again it could be of a different administrative domain. The system must be interoperable with the existing grid or distributed access control mechanisms.

This article is organized as follows: Related work is presented in section 2. In section 3 we construct a model of our

[*] This work is partly supported by the Région Rhône-Alpes RAGTIME project and the French ministry for research ACI-GRID project (`http://www-sop.inria.fr/aci/grid/public/`)

environment and use it to motivate our design goals. Section 4 explains our proposal of encryption key management. We expand our proposal in section 4.6 to make it suitable for distributed computing and to eliminate single points of failure. Section 5 summarizes the services that are required for our proposal and their required capabilities. A discussion of the drawbacks and the advantages of the presented system follows in section 6. We conclude and summarize our results in section 7.

## 2. Related Work

Current secure storage systems mainly provide one or both of the following features: encrypted storage and secure communication between storage device and the user.

*CFS* [2] provides encrypted storage at directory level for a local file system. However no specific file sharing mechanism is provided, so that users must individually provide the decryption keys if they want to share some of their files. Therefore CFS 'as is' is not suited to be used in distributed environments with complex access control policies.

*SFS* by P. Gutmann[1](As stated in [6], this is completely unrelated to D. Maziéres SFS), provides encrypted storage at disk volume level. An interesting feature is the emergency key recovery mechanism which employs Shamir's secret sharing scheme [14] to distribute key shares to trusted escrow agents. As the system seems to be unsupported since 1996 and lacks encryption granularity and platform independence it is not usable in distributed or grid environments.

*Cepheus* [6] enhances the functionalities of SFS[10] by D. Mazières et al. to provide encrypted storage and communication as well as distributed file sharing mechanisms. File sharing within a user group is managed by symmetric group keys that are stored encrypted with the group members public keys on a group database server. This may cause coordination problems in a distributed environment where groups contain members from different administrative domains. If such a group is managed by multiple authorities from the different domains and membership changes dynamically, keeping the group database up to date could quickly become an administration nightmare.

*SFS* (Also completely unrelated to both other homonyms) by J. Hughes et al. [7] provides encrypted storage on single file level and key management for decentralized access. The data is to be encrypted by the data producer and to be decrypted by the data consumer, therefore eliminating the need for encrypted communications. Accessing is done with the help of a trusted group server. The file encryption keys are encrypted with the group servers public key and stored in the file header together with an access control list (ACL). If a user wants to access an encrypted file he will have to forward the header to the group server, who will then use the ACL to determine if the user has access to the file. The problem we see with this architecture in distributed or grid environments is the following: If the access permissions (i.e. the header) of a file are changed while a storage site with a replica of this file is offline, this replica will still have the old access permissions when it comes online again, therefore leading to undesirable inconsistencies in the overall access permissions.

## 3. Motivation

In this section we will first present some terminology we use in our contribution, then we will give a short overview of current access control models. From those we construct a generic classification of access control decisions with regard to the management of encryption keys and use this classification to determine the requirements for key management systems in distributed or grid environments.

### 3.1. Notation

**Definition 1 (Access control)** *The goal of* access control *is to regulate the relations between users and resources by defining which resources can be accessed by which users.*

**Definition 2 (Objects, subjects and administrators)** *The users will be called* access control subjects *(or short* subjects*) and the resources will be called* access control objects *(or short* objects*). The entities that are authorized to make access control decisions for some object will be called* object administrators *(or short* administrators*).*

As encrypted storage only makes sense for data (in contrast with other resources as computing power and storage space), we also limit access control objects to files in the scope of this paper.

### 3.2. Access control models

To create our system we have to make some assumptions about the environment in which our system is to be used. More specifically we need to take into account the access control model that governs the environment. The entity that administrates access rights also has to provide access to the corresponding decryption keys and the key management scheme has to take into account all possible access control decisions that can be taken by the access right administrators.

We will examine the following access control models:

[1] http://www.cs.auckland.ac.nz/∼pgut001/sfs/

- A simple discretionary access control (DAC) model, where the owner of a file decides who gets access and who doesn't.
- A mandatory access control (MAC) model, where subjects and objects are labeled with security levels. Subjects may read objects of their security level and below and may write to their security level and above.
- A role based access control (RBAC) model, where permissions are grouped into *roles* based upon the jobs to be performed within the system. Subjects are assigned the permission to take certain roles and to use the associated access permissions.

For more specific discussion of the presented access control models see [11].

### 3.3. Classification of access permissions

From the access control models presented in 3.2, we obtain the following use cases for file access permissions:

- *Simple access permission:* The subjects and the objects of the permission are known at the moment it is issued and do not change dynamically. This is typically the case in a DAC model. An example is the administrator of a UNIX system who sets the ownership of a file with the *chown* command. Figure 1 illustrates this setting.

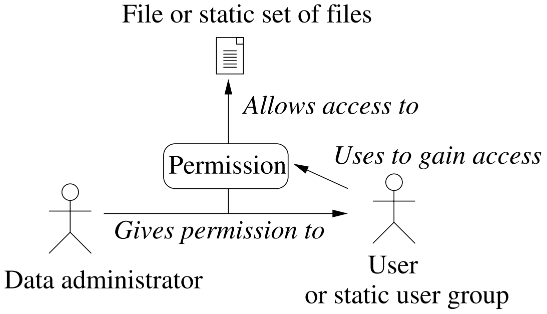

**Figure 1. Simple access permission scenario, subjects and objects of the permission are known and static**

- *Object set permission:* The subject of the permission is known at the moment it is issued and does not change dynamically. The object of the permission is a set of files with dynamically changing content. This is typically the case in MAC and RBAC models. An example is a government employee who is given a security level to access a set of classified files on his agencies computer system. New files will be labeled with a security level which will determine if they are accessible to that employee. Figure 2 exhibits this scenario.

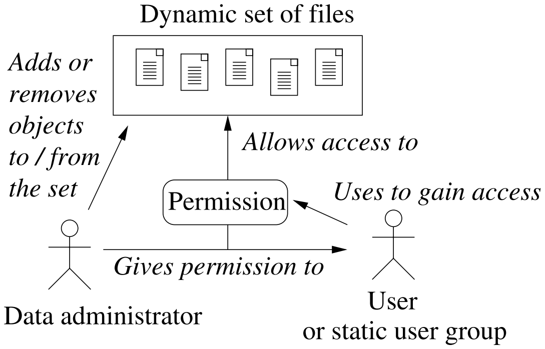

**Figure 2. Object set access permission scenario. Administrators of the object set may dynamically change its contents.**

- *Subject group permission:* The objects of the permission are known at the moment it is issued and do not change dynamically, the subjects of the permission form a group with dynamically changing membership. Thus the individual subjects of the permission are not known at the moment it is issued. This case can be either set in a DAC or MAC or RBAC model. An example out of an RBAC model would be a role that is assigned some specific file access permissions (e.g. all subjects who have the permission to use the role *programmer* are authorized to write to the object *program_code.c*). We show an illustration of this case in figure 3.
- *Permission concerning a subject group and an object set:* Both subjects and objects of the permission are dynamically changing and are not individually known at the moment the permission is issued. This can typically happen in a RBAC model. An example would be a role *medical researcher* that is assigned the access permission to a set of files *anonymized patient data*. The set can be changed independently of the role definition by adding new file or removing existing ones. The permission to use the role can also be reassigned without changing the role itself. This final case is illustrated in figure 4.

Please note here that all of these permission types may of course contain conditions that are evaluated at run-time and may therefore lead to a denial of access for an otherwise authorized subject (e.g. in the permission "the subject

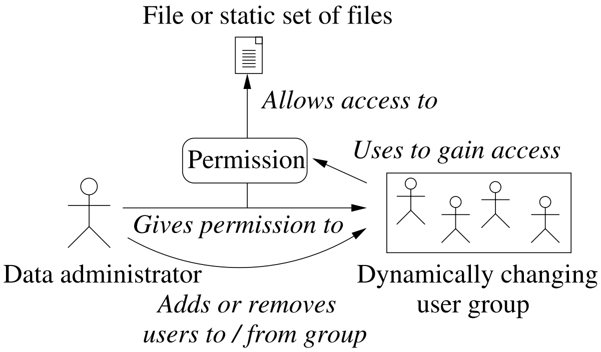

**Figure 3. Subject group access permission scenario. Administrators of user group may dynamically change its members.**

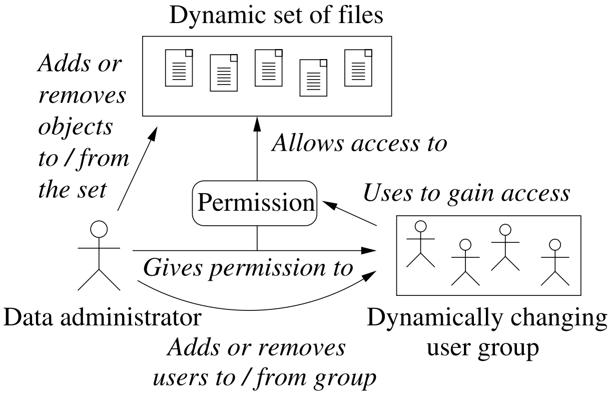

**Figure 4. Subject group to object set access permission scenario. Membership in the subject group and the contents of the object set may be dynamically changed by the administrators.**

authentified as *Alices doctor* may access the object *Alices health record* if he is logged onto a secure terminal", being logged onto a secure terminal would be a run-time condition). However this does not change the main conditions by which above classification has been done, which is whether subjects or objects are individually known when the permission is issued. As those run-time conditions are evaluated by the access control system which we intend to use, we have not taken them into account any further.

### 3.4. Design goals

We pretend the four cases presented in section 3.3 cover all possible file access control decisions in all three access control models described in 3.2. Our global goal is to create a scheme that allows the storage of, and access to encrypted data on distributed, heterogeneous storage devices.

- We want our proposal to be interoperable with most access control mechanisms without functional changes. That means our key management service should not require a change of the environment's access control model. It should be usable as an add-on and should be able to run in mixed settings where a part of the data is not stored encrypted.
- For reasons of vulnerability and to spare bandwidth we want to minimize the use of third parties. In cases where a third party is used, we want to minimize the impact in the event it becomes corrupted.
- Subjects should have the responsibility of decrypting the objects. It may first appear to be an advantage to do the decryption independently from the subject and to securely transfer the decrypted data to it. Revoking access permissions would be more simple, since the subjects would not have the encryption keys of objects they are authorized to access. We decided against such a solution since the decryption service would be an access bottleneck and a dangerous source of vulnerabilities, as it would have access to all decrypted data.
- We want to avoid a centralized administration of decryption keys. Centralized services do not scale well in distributed and grid environments, often becoming bottlenecks and single points of failure. There must be redundant possibilities of accessing the encryption keys to reduce the risk of a denial of access and to provide for a good scalability.
- The user should not have to trust a single key administration service. Since the entity providing this service will probably be outside of the users administrative domain, he has no possibility to control if the service runs correctly and is not corrupted. Therefore a single key administration service should only have access to key shares and not to complete keys.

## 4. Key management

Our key management scheme has to take into account the four access scenarios we presented in section 3.3:

### 4.1. Simple access permissions

In this situation we have several possible situations:

1. The subjects of the permission are online and no conditions that have to be evaluated at run-time are part of the permission.
2. At least one of the subjects of the permission is offline, and no conditions that have to be evaluated at run-time are part of the permission.
3. The permission involves conditions that are to be evaluated at run-time.

In the first case we choose to avoid using a third party and transfer the keys to the subjects over a secure channel (this may be an encrypted mail, or a direct connection secured by SSL, SSH, IPSec or another secure communication protocol).

Figure 5 illustrates this procedure.

In the second case, one could consider storing the keys encrypted with the subjects public key together with the objects. However this will lead to an increase of the objects meta-data, which might be impractical if the number of subjects having access to the object becomes large. Therefore it seems to be a more scalable solution to store these keys on a key server. This server must be capable of verifying the access permissions and granting access to the keys if an authorized user requests them. To avoid making the key server a single point of failure, we propose duplication and secret-sharing mechanisms in section 4.6 (we still use the singular term *key server* to avoid complicating the expressions).

In the third case the key server becomes inevitable, as it has to check the run-time conditions, before granting access to the keys. Figure 6 illustrates the procedure of accessing an object with the help of a key server.

### 4.2. Object set permissions

With object sets coming into play, the key of an individual object is only known from the moment the object is added to the set. Furthermore an entity adding objects to the set may not know which subjects have access permissions for that set.

Therefore the keys of objects being added to the set can not be directly distributed to the subjects. They need to be stored on a key server by the object administrator as described in the previous section and illustrated in figure 6.

### 4.3. Subject group permissions

The subject group permissions again raise the same problem: the administrator can not know the individual subjects concerned by the access control decision. They may change after the decision has been made, as new members are added to the group or old ones removed. Therefore the encryption keys can not be distributed directly and again they have to be stored on a key server.

When a subject of a group permission presents himself to a key server he must prove that he indeed belongs to that group. How this is done depends on the used access control system. If a certificate based access control system like [13] is used, this can be done without contacting another service, thus reducing the required network traffic. The key server can then decide if the group permissions allows access to the requested keys.

### 4.4. Granting permissions to subject groups and object sets

We now consider a case of permission where the objects and subjects are dynamically changing.

At the creation of the permission all encryption keys of files belonging to the object set have to be stored on the key server. Depending on the access control system the permission might need to be registered at the key server, to allow verification of key access requests.

If the administrator add new objects to the set he must also provide the decryption keys to the key server.

Should new subjects be given group membership the key server must be enabled to verify this new status. Again if we have a certificate based access control system like in the example in 4.3 this can be done by issuing a group membership certificate to the subject.

### 4.5. Revoking permissions

When an access permission is revoked, the subject may be able to keep copies of the unencrypted data or even the keys he had access to. Even if this is not the case, there is no way to prevent him from disclosing the content of the data after he accessed it.

However it may be required, to prevent former subjects from changing the object or reading new versions of it. There are several ways to prevent this. The first is to re-encrypt an object, when an access permission is revoked and to update the key on the key server. This is cumbersome and consumes a lot of computing power.

If the storage sites have an access control mechanism, we can simply deny the access to the encrypted data on the storage site and leave the encryption key unchanged. The subject may still get access to the data, if he gets administrator access to the storage device.

An intermediate way would be to do a lazy update, where re-encryption is delayed after a permission change, until the concerned object is actually changed.

We believe that there is no best solution to the revocation problem. Therefore the system should provide the administrators with the option to choose which of the three presented solutions he wants to use. As our approach is designed to be used in concert with a classical access control

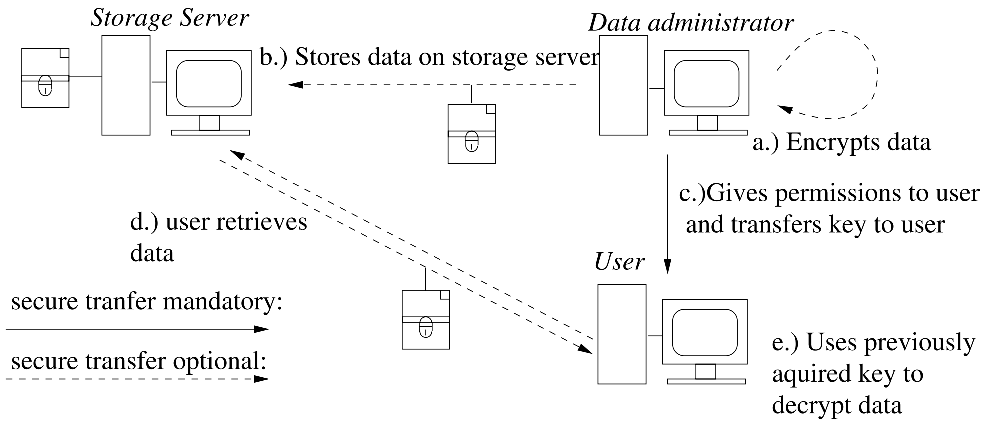
Storage Server
Data administrator
b.) Stores data on storage server
a.) Encrypts data
c.)Gives permissions to user and transfers key to user
d.) user retrieves data
User
secure tranfer mandatory:
secure transfer optional:
e.) Uses previously aquired key to decrypt data

**Figure 5. Access procedure for a permission with no conditions evaluated at run-time, where subjects and objects of the permission are known when it is issued.**

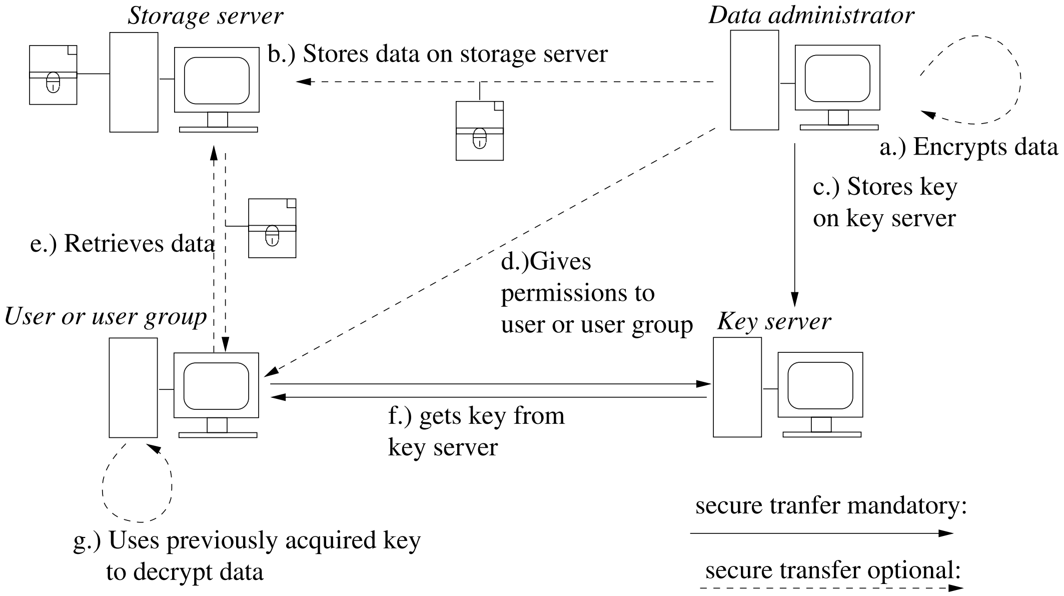
Storage server
Data administrator
b.) Stores data on storage server
a.) Encrypts data
c.) Stores key on key server
e.) Retrieves data
d.)Gives permissions to user or user group
User or user group
Key server
f.) gets key from key server
g.) Uses previously acquired key to decrypt data
secure tranfer mandatory:
secure transfer optional:

**Figure 6. Access procedure when the decryption keys of the accessed object are managed by a key server**

mechanism, the second option is to be used to minimize consumption of computing power.

### 4.6. Avoiding single points of failure

In a distributed environment with lots of requests the key server may become a communication bottleneck and even a single point of failure. If a single key server holds the keys to all objects in our environment, it will be a prime target for attackers willing to compromise the system or more simply to run a denial-of-service attack. It is therefore reasonable to operate a network of distributed key servers.

We present two independent but interoperable proposals for distributing the key servers. The first one is intended to increase availability through replication and clustering of the keys. The second proposal aims to reduce the risk of key disclosure in the case a server becomes compromised.

**4.6.1. Increasing availability** The key server holds a set of keys and some associated meta-data that allows it to associate the keys to encrypted files and to check access permissions. However there is no reason why a single key server should hold all keys or why a key should be stored on one single key server. Therefore keys are distributed and replicated on different key servers.

In many application scenarios a natural way of doing this distribution exists. An obvious solution would be to use the semantic meaning of the objects they encrypt. For example if we distribute keys to medical data based on the pathology they describe subjects are more likely to access several objects in one set than objects of several different sets. Figure 7 shows an example for a semantic distribution of medical data.

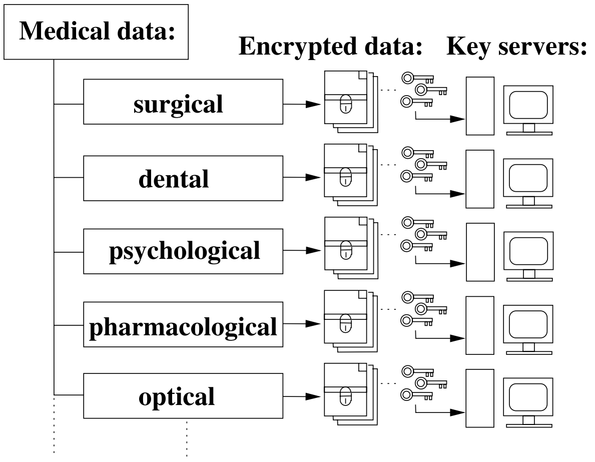

**Figure 7. Example of semantic distribution of medical data. The keys of different data sets are stored on different key servers.**

Therefore by distributing encryption keys on different key servers the load is spread and multi-object requests normally do not require contacting more than one key server.

If no object sets exist or a finer distribution of the objects is required, this can be done by applying a hash function to the unique object identifier and creating (sub-)sets of the objects by the result of this hash. This hash could be as simple as taking the first characters of the object identifier or more complex, if a more balanced distribution is desired.

The remaining problem is how subjects can find the right key server for the objects they want to access. The most natural way here, is to store this information together with object meta-data on the storage site. As seen in figure 6, the subject has to obtain the encrypted object on the storage site before contacting the key server. So if we store the location of the appropriate key servers in a data-base on the storage site together with the encrypted data, they can be easily found by a subject accessing the data.

Depending on the degree of availability we whish to obtain, distribution of the key server alone may not be satisfactory. The failure of one key server would still mean access to the object sets it manages would not be possible. Therefore the key server should be replicated mirroring its content to several identical servers.

**4.6.2. Reducing vulnerability** In grid computing, the entities providing storage space are typically anonymous to object administrators, as the grid middleware hides the complexity of storage space allocation from them. In the same way, the middleware should hide the management of different key server from the object administrator. Therefore we present a scenario where the object administrators have limited trust in the key server, and do not entrust the complete key information to a single one.

Classical secret sharing mechanisms like [14] can be used to divide single keys between several key server. An arbitrary number $m$ is chosen at the time a key is to be stored. Then a number of key shares $n \geq m$ is produced from that key. These shares are distributed to $n$ different key servers. If an authorized subject wants to recover the key, he has to gather $m$ different key shares from the key servers. There is no risk of key disclosure, as long as less than $m$ key servers are compromised. Figure 8 illustrates data access with the use of key shares.

Key sharing combines well with key server replication. Note that *any* combination of $m$ different key shares will enable a subject to reconstruct the key. Thus an object administrator wishing to guaranty access to the key even if $b$ key servers break down, simply has to chose the number $n$ of key shares he generates as $n = m + b$ and distribute them on $n$ different key servers.

If the number of objects that are handled within the system is relatively small, key sharing may make the distribution of keys on servers by object sets unnecessary. However both still combine without problems, if we distribute key servers by object sets first and then by different share servers.

## 5. Implementation Issues

We will now summarize services that have to be created to implement the proposed scheme and we describe

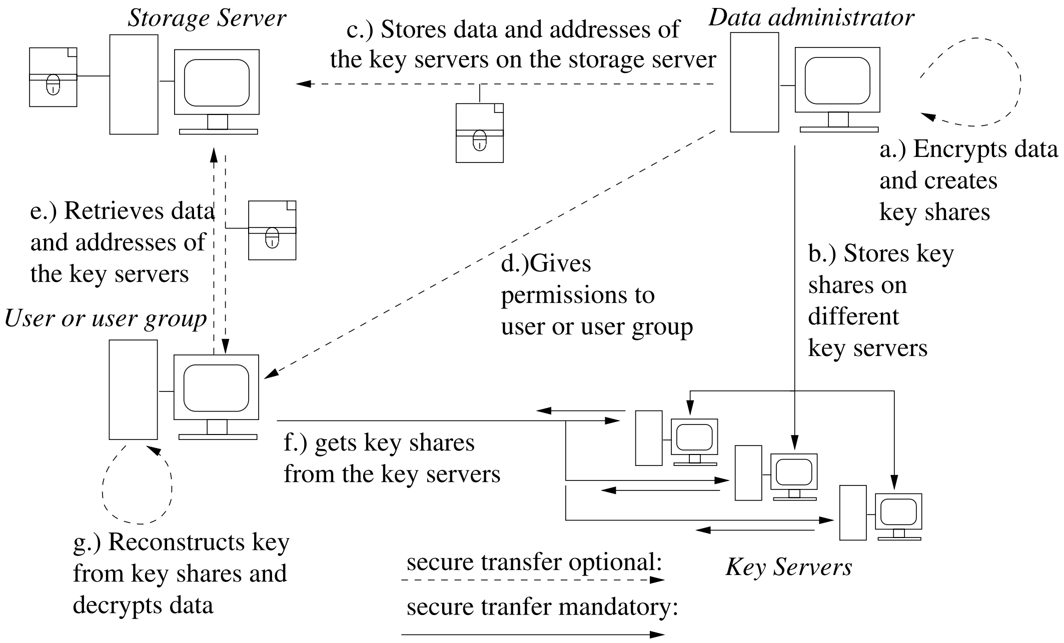
Storage Server
Data administrator
c.) Stores data and addresses of the key servers on the storage server
a.) Encrypts data and creates key shares
e.) Retrieves data and addresses of the key servers
d.)Gives permissions to user or user group
b.) Stores key shares on different key servers
User or user group
f.) gets key shares from the key servers
g.) Reconstructs key from key shares and decrypts data
secure transfer optional:
secure tranfer mandatory:
Key Servers

**Figure 8. Access procedure when shares of the decryption keys are stored on different key servers.**

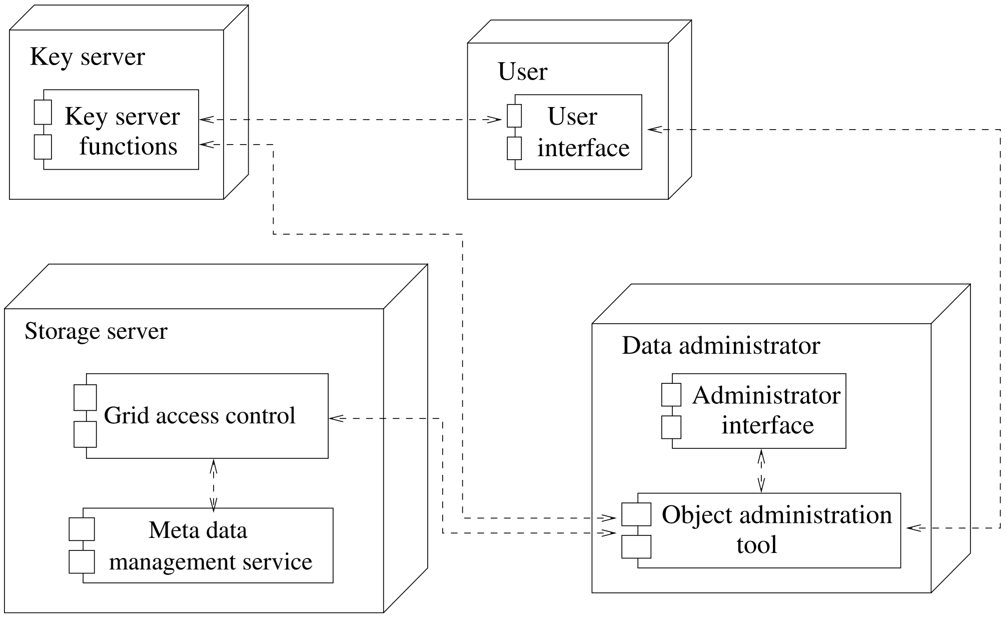

**Figure 9. Deployment diagram of the services required for the key management architecture**

the functionalities these services must offer. Figure 9 shows the deployment of those services.

### 5.1. Key servers

A key server can authentify itself to any administrator, using a certificate signed by a trusted authority and a challenge-response protocol.

It operates a database to store and retrieve the key shares. This database is a local service which requires a password or similar authentication to access. It should not be accessible by other services than the key server.

The key server can authentify users requesting key shares with the same mechanism it uses to authentify itself. It is also able to make access control checks to determine if a request for a key share should be granted. This key share access control must be coordinated with the general data access control services.

Revocation of access rights is to be handled by 'push'-type key server updates (i.e. the entity revoking an access right actively sends the information to the key servers). Key servers actively update their access rights in a 'pull'-manner after an offline period. This is done by contacting other key servers. To make this updates fault tolerant and to prevent deliberately wrong updates coming from a corrupted key server multiple key servers are contacted and only permissions that are coherent on a majority of the key servers are updated.

The key servers provide secure communication services to transmit key shares and updates of access rights over insecure networks. Either TLS/SSL, SSH or IPSec shall be used for those.

Figure 10 summarizes the functions of a key server.

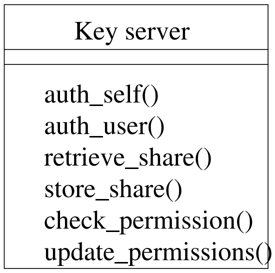

**Figure 10. Functionality of a key server.**

## 5.2. The object administration tool

In order to store objects, their administrator needs a special tool that will encrypt the object, create the key shares and distribute them over different key servers.

Optionally the object administration tool can add a digital signature to the encrypted object to ensure the object integrity. The administration tool shall rely on certification authorities to provide the public keys needed to check those signatures.

The object administrator can prevent denial of access to objects by creating multiple replicates of the objects on different storage sites.

Depending on the use of the objects, different symmetric encryption algorithms are recommendable. In software, *stream ciphers* like ARC4 or SEAL are at least twice as fast as *block ciphers* like the AES[2]. However, if we want to change a part of an object encrypted with a stream cipher, we will have to decrypt the whole object, whereas block ciphers in ECB (electronic code book) mode permit to decrypt and change single blocks of the data separately. For more details on cipher algorithms refer to [12].

The tool is local and controlled by the object administrator, since it handles the unencrypted versions of the object and the object administrator needs to trust it.

Figure 11 summarizes the functionality of the object administration tool.

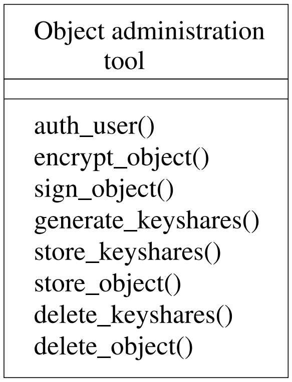

**Figure 11. Functionality of the object administration tool.**

## 5.3. The user interfaces

For ease of use all services shall provide simple interfaces and hide most of the complexity of their operations to the users. Both the administrator and the data access interface shall be single sign-on (i.e. a user should only have to authentify himself once during a session).

In the case of the administrator interface, the user should only have to select between storing and deleting data, as well as granting and revoking access rights. The different configuration options (number of key shares, used ciphers,

[2] In a test with the Crypto++ library on a 1.9 GHz Pentium 4, ARC4 encrypted at a rate of 24 MB/s, SEAL at 60 MB/s and the AES with a 128-bit key at 10 MB/s

replication etc) shall be accessible to him, but default options shall be provided.

The data access interface should hide the different steps of an access operation, such as contacting the key servers, reconstructing the keys, downloading the data and decrypting it. From the point of view of the user it should only perform three distinct functions: Authentication of the user at the start of a session, update of known permissions, searching and retrieving data. The search and retrieve function will temporarily pass control to other distributed or grid services to perform the functions unrelated to security. The access interface shall store retrieved encryption keys locally to speed up further requests for the same data. This storage shall of course be encrypted and protected by the initial authentication at the start of a session.

Figure 12 shows the functions of both administrator and user interfaces. The administrator interface will use the object administration tool for functions concerning the encrypted storage and key management. Both will also use other services (i.e. the access control service, data storage service) provided by the distributed environments infrastructure.

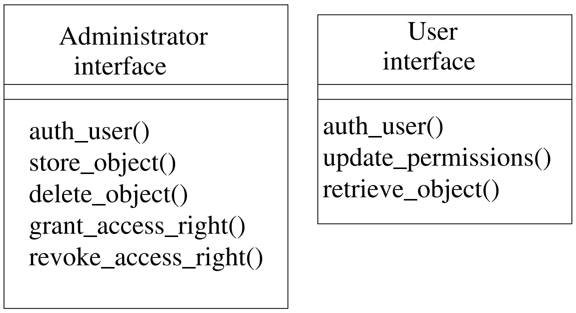

**Figure 12. Functionality of the administrator interface and the user interface.**

### 5.4. The access control system

Depending on the access control system, modifications may be necessary to keep the key servers access permissions up to date. In our certificate-based access control example from section 4.3, the key server must know the types and structures of the certificates used and must be able to verify their signatures. Thus no modification of the access control system itself is necessary.

## 6. Discussion

In this section we discuss the drawbacks and advantages of the proposed system, with special attention to distributed and grid computing environments.

The drawbacks of the system come principally from the functions that have to be performed by the key servers.

- Contacting different key servers before being able to decrypt a file slows down the access procedure.
- As the key servers are partially trusted third parties, an attacker may gain access to encryption keys if he manages to corrupt multiple key servers. Setting the number of key shares needed for reconstruction high enough and careful administration of the key servers should minimize this risk.
- To provide links to corresponding key shares some amount of meta-data needs to be stored together with encrypted data. However most grid environments require some meta-data for data storage anyway. Therefore this should only be a minor administrative problem.
- The key servers have to manage access to the key shares and thus indirectly the access to the encrypted files. Therefore certain access control mechanisms may make it necessary to update the key servers access control information, every time an access permission is issued or changed. However all current access control mechanisms adapted to distributed or grid environments [13], [1], [9], [3] allow decentralized access permission checking. We therefore believe that this problem will only appear with ill-chosen access control mechanisms.

The system has the obvious advantage of eliminating the need to trust the storage servers. Additionally we have found the following advantages to our design:

- Aside the transfer of the meta-data to locate corresponding key servers, the design requires no additional interaction with the storage servers.
- The size of the meta-data required on the storage servers is minimal (addresses of the key servers).
- The system is designed to be interoperable with common access control mechanisms. It can be used optionally and does not exclude unencrypted storage of less sensitive files on the same system.
- Together with distributed or grid access control mechanisms the key-server architecture allows for distributed and fine grained access control decisions from different independent administrative domains.
- The key sharing mechanism makes the system tolerant to limited key server breakdowns. It also ensures that

the corruption of a single key server will not create a major security breach.

We think that the advantages provided by our architecture outweigh the loss of performance it will cause. We intend to implement our proposal to measure effects of the additional network traffic it creates. However one must keep in mind that it is always difficult to evaluate performance loss against additional security gained. This is also the main reason why our design enables administrators to choose which tradeoff is the best for their needs.

## 7. Conclusion

We have presented the problem of managing keys for encrypted data storage in distributed or grid environments. We showed that access control management in such environments has to deal with problems that are not treated by classical encrypted storage systems. These problems come from the fact that access permissions may involve users and data from different administrative domains. We detailed the problems arising from user groups and data sets administrated by multiple authorities from different domains.

Our key management architecture is adapted to those problems and was designed to be easy to add onto common distributed or grid access control systems. It can manage variable granularities of access permissions and can be parameterized to fit the security needs of different data administrators.

We plan to implement this key management architecture for the MEDIGRID[3] project, using the hDSEM[5] and the grid access control system proposed in [13]. With this implementation we will be able to measure the delay caused by contacting the key servers. Another interesting question would be to determine reasonable parameters for the settings of our system (number of key shares, number of key servers, number of data replicas etc).

## References

[1] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, A. Gianoli, K. Lörentey, and F. Spataro. VOMS, an authorization system for virtual organizations. In *Proceedings of the 1st European Across Grids Conference*, 2003.

[2] M. Blaze. A cryptographic file system for UNIX. In *ACM Conference on Computer and Communications Security*, pages 9–16, 1993.

[3] D. Chadwick and A. Otenko. The permis x.509 role based privilege management infrastructure. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, 2002.

[4] T. Dierks and C. Allen. The TLS protocol version 1.0. Technical report, The Internet Engineering Task Force IETF, 1999. http://www.ietf.org/rfc/rfc2246.txt.

[5] H. Duque, J. Montagnat, J. Pierson, L. Seitz, L. Brunie, and I. Magin. An architecture for large scale and high performance medical imaging applications. Available from http://hectorduque.free.fr/recherche/tdPapers.html.

[6] K. Fu. Group sharing and random access in cryptographic storage file systems. Master's thesis, Massachusetts Institute of Technology, 1999.

[7] J. Hughes, C. Feist, S. Hawkinson, J. Perrault, M. O'Keefe, and D. Corcoran. A universal access, smart-card-based, secure file system. In *Proceedings of the 3rd annual Atlanta Linux Showcase*, 1999.

[8] IPSec Working Group. IP security protocol (ipsec). Technical report, The Internet Engineering Task Force IETF, 2002. http://www.ietf.org/html.charters/ipsec-charter.html.

[9] K. Keahey and V. Welch. Fine-grain authorzation for resource management in the grid environment. In *Proceedings of the 3rd International Workshop on Grid Computing*, 2002.

[10] D. Maziéres. Security and decentralized control in the SFS global file system. Master's thesis, Massachusetts Institute of Technology, 1998.

[11] R. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[12] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition*. John Wiley & Sons, 1995.

[13] L. Seitz, J. Pierson, and L. Brunie. Semantic access control for medical applications in grid environments. In *Euro-Par 2003 Parallel Processing*, volume LNCS 2790, pages 374–383. Springer, 2003.

[14] A. Shamir. How to share a secret. In *Communications of the ACM*, volume 22, pages 612–613, 1979.

[15] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH protocol architecture. Technical report, The Internet Engineering Task Force IETF, 2002. http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-13.txt.

[16] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

[3] http://creatis-www.insa-lyon.fr/MEDIGRID