View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com

ScienceDirect

Journal of Computer and System Sciences 74 (2008) 243–254

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Stateless data concealment for distributed systems

Rachid Anane [a,*], Sukhvir Dhillon [b], Behzad Bordbar [b]

[a] *Department of Computer and Network Systems, Coventry University, UK*
[b] *School of Computer Science, University of Birmingham, UK*

Received 15 June 2006; received in revised form 31 October 2006

Available online 24 April 2007

Abstract

With the growing number of Web applications and their variety, the need to prevent unauthorised access to data and to ensure data integrity in distributed systems has led to an increasing reliance on encryption. Within a node, a typical encryption process operates at file or directory level and applies indiscriminately one algorithm to its data. In this paper, a scheme is proposed whereby secrecy is achieved through file data and file location concealment, within a client–server distributed system. This involves the division of a file into fragments, their encryption and compression, the random allocation of these fragments to the nodes, the generation and transcription of metadata for reconstructing the original file, and finally the deletion of both the original file and its metadata from the local node. A prototype of the scheme was implemented and evaluated in terms of the performance of the distribution and recovery process.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Distributed system; Data concealment; Security; Encryption

1. Introduction

Research into the prevention of unauthorised access to data has acquired more urgency as a result of the increase in the number and the variety of software applications, and the widespread adoption of wireless environments. The requirements for flexibility and availability in distributed systems have also been motivating factors in the design of new architectures. This concern has led, in particular, to designs where server and storage operate independently with a heavy reliance on encryption [1,2]. This represents a significant departure from the traditional centralised storage approach, where the design of secure systems follows the model of a reference monitor.

In a traditional model a centralised file server controls the file system and access to files is managed by means of access control lists (ACL), a feature that identifies discretionary access mechanisms [3]. The system is usually managed by an administrator or someone who has super user privileges; the access control lists can only be altered by the owner of the file or the super user. Any executing process initiated by the user inherits his/her user rights and access privileges. The implication of this mechanism is that any program executed by the user, or on behalf of the user is capable of initiating operations, which may be harmful. For example the executing process could delete or modify

* Corresponding author.
E-mail addresses: r.anane@coventry.ac.uk (R. Anane), s.dhillon@cs.bham.ac.uk (S. Dhillon), b.bordbar@cs.bham.ac.uk (B. Bordbar).

0022-0000/$ – see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcss.2007.04.007

files and permit other users to access files by altering the associated file access control list. The restriction of access to files, based solely on the identity of the users, leaves them open to various forms of attack and in particular to Trojan horse attacks. Moreover, an ACL is suitable for static configurations as dynamic access controls are difficult to define, while modification of an ACL is awkward and presents a serious impediment to scalability. On a wider scale, a reference monitor provides security of access to objects at the server level, and does not address the confidentiality and integrity of the data when it is transferred between nodes. Another characteristic of an ACL is that the storage of data is tightly coupled with access control policies.

One fundamental limitation of discretionary access control mechanisms is their inability to enforce a logical classification of information based on the level of the protection it requires. Solutions to this problem should enable the user to discriminately group files on the basis of the level of protection they require, and should also prevent the targeting of secure files. In order to address this issue, many encryption schemes were presented as a way of enforcing secrecy, authentication and authorisation [4]. Encrypted files whose access is controlled solely through the use of discretionary access control systems are still vulnerable to attacks by malicious software, especially when weak ciphers are used. Another issue related to encryption is granularity; the prevailing models apply one encryption algorithm at file level, identically to all its elements**.**

Novel schemes have been introduced as a way of addressing security issues in a distributed environment. These are motivated by the lack of flexibility of access control policies and the need to ensure confidentiality and integrity of data when transiting between nodes. These file systems operate either at user level or the kernel level. The cryptographic file system presented by CFS [5] operates as a user level NFS server [6]. The encryption of the directories determines the specification of the cipher and keys. This user level implementation makes it a highly portable file system, at the expense, however, of performance. A related issue refers to the lack of transparency with respect to users and operating system since special commands are required for the deployment of the scheme. Furthermore, the authentication function is not easily supported.

In contrast to CFS, TCFS [7] adopts a different perspective; it is implemented at kernel level as an NFS client. The system does not require special commands and exhibits greater transparency with respect to users. The integration of the scheme with Unix manifests itself in the greater reliance of the system on the Unix authentication system, with login passwords acting as user keys. This tight coupling is also evident in the way the keys are stored in a Unix database. Although many benefits accrue from this integration the system is however vulnerable to crypt-analytical attacks. At the other extreme of the spectrum, the architecture presented by Reed et al. [8] marks a significant break with the classical model, where access to data is mediated by a file server. Storage is no longer a passive entity but is seen as an agent actively engaged in mutual authentication with clients. This approach shifts the focus to the authentication mechanism, a central component in the implementation of network-attached storage.

Harrington et al. [9] present a scheme based on asymmetric encryption. Data is encrypted and decrypted with a private/public key pair; these two procedures correspond to the read and write operations, respectively. Unlike a traditional centralised scheme, where access control is secured by means of capabilities, this approach promotes distribution and the semantics of the keys are clearly defined at the point of use rather than at the server level. An encrypted file within this scheme is however still visible and can be accessed by exploiting the flaws inherent to discretionary file access. It can be subjected to a crypt-analytical process.

Most of the proposed systems, however, are an attempt at grafting a layer of security on top of existing network operating systems. In addition, the schemes presented in the literature conform to the view that confidentiality is addressed mainly by encrypting and decrypting files, at the client side only [10]. They share the feature that files are seen as integral entities; the main focus is on the encryption of data so that it is difficult to access it either on disk or during transfer. Most of these schemes are closely coupled to the operating system or the network. The scheme proposed in this paper is an enhancement to a distributed system, such as the Web, operates at a higher level and is implemented at user level. Within this scheme, file location concealment is achieved in addition to file data concealment. This is realised by the fragmentation of selected files, their encryption and their distribution across a network of nodes, and the removal of all related structural information such as metadata.

This remainder of the paper is organised as follows. Section 2 presents the design goals and the architectural framework of the system. Section 3 gives an outline of the implementation and the underlying mechanisms for securing entities and communication. Section 4 covers performance issues in terms of various parameters. Section 5 puts the scheme into perspective through a comparison with other implementations. Further work is presented in Section 6, and Section 7 concludes the paper.

2. Architecture of the proposed system

The main objective of the proposed system is to enable users to store and access sensitive information securely. The scheme complements and enhances the functionality of a traditional discretionary access based file system, by providing a layer of security on top of existing network operating systems and by taking advantage of the distribution of nodes.

2.1. Design goals

The scheme is part of an effort at overcoming the shortcomings of discretionary access at a relatively low cost [11]. It is independent of the operating system and its file system. More specifically, the main aims of the proposed scheme include:

(1) Overcoming the limitations of discretionary access mechanisms.
(2) Using encryption as a way of storing and accessing files in a secure manner.
(3) Achieving total concealment of files and statelessness through migration or removal of metadata.
(4) Ensuring secure communication in client server interactions.
(5) Presenting a system at user level with a high level of transparency with respect to operating system and network.
(6) Offering a generic mechanism for secure access to data in distributed systems such as the Web.

A central feature of the proposed scheme is the enhancement of the functionality of the host operating system, at user level and with minimal intrusion. The scheme ensures that data is in plain text only when it is used at the client node. Otherwise the data is in encrypted form and the local files hold partial information only. The ultimate aim of the scheme is to support file data and file location concealment. A number of factors have presided over the design of the system:

- *Security and authentication*: data should be protected against malicious access. In a file system patterns should not be easily detected. Since the metadata of files contains sensitive information, it should also be protected or hidden completely. Data integrity in a distributed system should be preserved during transfer. In addition, the whole scheme should be supported by authentication mechanisms in order to secure client server communication.
- *Performance*: the system should have acceptable performance, within the constraints of the trade-offs between secrecy and availability. A minimal load on the distributed system should be generated by the deployment of the scheme. The selection of appropriate encryption mechanisms should minimise computational overheads. For example, since symmetric encryption schemes are less onerous than the asymmetric ones they should be selected whenever possible.
- *Usability*: the system should present a simple user interface for file management, with implementation details hidden from the users, in order to provide greater support for transparency. Encrypted files should be treated in a uniform manner by the underlying system, and key management should be an integral part of the secrecy and authentication mechanism. The system should support user mobility in its symmetric behaviour; and centralisation is circumvented by the ability of clients to act as file servers.
- *Portability*: the scheme should be implemented at user level by incorporating existing user interfaces. Files or fragments of files should be managed by the underlying system without recourse to special commands. The scheme should be implemented in a platform independent language and should be portable across different operating systems.
- *Flexibility*: the scheme should support a finer granularity of data than classical schemes. In particular it should be possible to encrypt part of a file, at block level rather than at the file or at directory level. The random distribution of the fragments of a file to nodes should be performed asymmetrically. Encryption should be performed at the client level only and servers should act as repositories.

2.2. Concealment process

While most systems are concerned with data encryption at a specific node the proposed scheme exploits network connectivity for enhanced secrecy in order to ensure some continuity of service. It utilises encryption and the distributed nature of the system to achieve *file data concealment* and *file location concealment*. Thus file data concealment is realised through extensive encryption of the data, or more specifically of fragments of the file. File location concealment involves the following steps:

(1) Identification of a file whose security cannot be ensured through discretionary access, at a particular node. The file is in plain text.
(2) Division of the file into plain text fragments.
(3) Encryption of fragments. Fragments are encrypted with different algorithms when possible. This achieves *file data concealment.* Fragments are compressed.
(4) Random selection of remote nodes and transmission of encrypted fragment to nodes, with one fragment per node where possible.
(5) Storage of fragment at remote node.
(6) Generation of file metadata, and in particular the lookup table to recreate the original file, and commitment to physical medium.
(7) Deletion of the original file and removal of metadata from the local node. This achieves *file location concealment*.

The lookup table contains information for retrieving and restructuring the original file, which is stored in the distributed system. This holds the location of each encrypted fragment, its identifier, the name of the encryption algorithm used for the fragment and the secret key used to encrypt it. The lookup table is used for downloading the fragments in a reverse process. Encrypted fragments are retrieved, decrypted and the original file reconstructed. The system does not assume any trust in the server or the communication network. Only the client is trusted.

2.3. Architectural components

The scheme is designed for two distinct groups of users: the ordinary users who wish to conceal files on the distributed system, and the administrators who are responsible for the management of users, the auditing of activities and the allocation and revocation of certificates. The scheme conforms to the client–server model and is symmetric in its behaviour. The architectural framework assumes a set of interconnected nodes, which run their own version of the scheme. Conceptually, the scheme is implemented by a number of modules in each node, which perform the following functions, as shown in Fig. 1.

- *Administration*. The main tasks involve the administration of the nodes running the scheme, keeping a secure record of all the administrators of the system. Access to the distributed system is reserved to trusted users. Within this module data repositories provide the means to store the list of administrators in a secure way on each node. The list of administrators is stored by converting the username and password of the administrators to a hash code using the Secure Hash Algorithm (SHA). As the hash cannot be converted back to the original username and

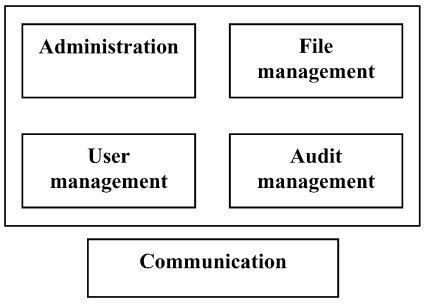

Fig. 1. Scheme components.

password, the administrators' details are stored in a secure form. This module is also responsible for generating a secret audit key and reading encrypted audit log files.

- *User management*. This provides for user registration and removal on the system and the ability to login and use resources on the system. A list of valid users should be held in a form, which would be difficult to interpret. Concern over secure user information parallels those identified for administrators. The list of users is also encoded and stored by converting the username and password to a hash code using the Secure Hash Algorithm (SHA).
- *Audit*. The main function in this module is to keep track of activities performed by users and to maintain secret audit keys. Logs are concerned with the recording of the occurrence of critical events during typical usage, i.e. audit logs can be created when the user or administrator logs into the system, when files are uploaded or downloaded, and when the server process is started or shutdown. However, as the audit logs may be subject to attack they should be protected through some form of encryption.
- *File management*. This encompasses encryption and decryption, compression and unpacking, uploading and downloading of files. It also provides a repository of cipher text. File processing defines a number of elements necessary to encrypt a set of plain text data, i.e. the size of the plain text datum and the names of the encryption algorithms used to encrypt the plain text data.
- *Communication*. A client component is responsible for the communication with a set of servers. Measures must be taken to ensure that untrusted processes are not able to interrogate the server process running on a machine. This requirement can be satisfied through the authentication of the processes that attempt to communicate with the server. In addition, the server can deal with multiple requests simultaneously i.e. concurrency is a design feature of the server.

2.4. Scheme deployment

Within the scheme, the distributed fragments belong to a distributed file, which is identified by the set of nodes that participate in the file location concealment. Each distributed file is associated with a specific set of nodes, and two distributed files may hold fragments on common nodes, thus leading to an overlap of the set of nodes, as shown in Fig. 2.

Users will be required to log on to the system every time they wish to use it. The administrator can also create groups of users. The system should, however, facilitate widespread access, i.e. the concept of "user mobility" should be integrated into system design [12]. User mobility will permit random access to any system node on the system and to gain access to confidential files. It implies that each node is required to hold a copy of the list of valid users and administrators. Although this creates some redundancy through the replication of data it avoids the need for a central repository, thereby increasing the overall reliability of the system and preventing resource bottlenecks.

File concealment requires the deletion of the original file and the newly created file lookup table from the local node. Since the deletion service offered by most file systems only replaces the pointer location in the file table, the file data is still present on the disk, and can be recovered through the use of disk scanning programs. Permanent and

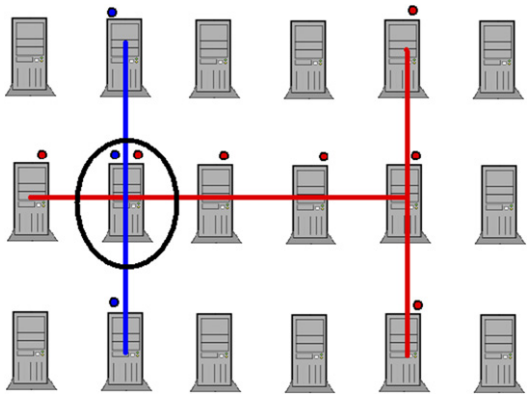

Fig. 2. Overlapping nodes.

complete deletion of files can however be achieved by incorporating file shredding programs such as the Linux *shred* program.

3. Implementation

The scheme has been implemented in Java, is platform independent and can run on a cluster of Unix or Windows machines. In the current version the scheme is deployed on a network of Linux servers, with the command line as the interface for applying the distribution process to a file.

3.1. Component interaction

The component modules of the system are made up of a number of essential as well as ancillary classes that perform various functions. For instance the administration module relies on two fundamental classes. The first is used for generating secret audit keys, reading encrypted audit log files and adding and removing users from the system. Creating keys for secure communication is the responsibility of the administrator prior to running the scheme, and is achieved through the use of the *KeyTool* facility, which comes as a standard program in the Java Development Kit. The second class acts as a repository and provides the means to store in a secure way the list of administrators on each node. Similarly, the communication module encompasses a class that deals with secure communication, which is achieved through the use of Secure Socket Layer (SSL) between client and server.

In the file management module, the size of the fragments is set to 1 K bytes. Two encryption algorithms are being used to encrypt the plain text: DES and Blowfish [2], whereas the compression of the encrypted fragment is performed by applying the Deflate algorithm. In order to identify the files relevant to the scheme unique file extensions are assigned to the files created by the system. File tables have the extension *ft* and each cipher text datum is given the extension *ct.* The extension *key* is used for secret keys, the extension *cert* for X.509 certificates and the extension *location* for the file which contains the list of every node which is running the scheme.

Figure 3 shows the interaction between the components of the scheme when a selected file is subjected to the process of concealment in remote servers. The Client is responsible for transmitting a fragment to the designated server.

3.2. Secure entities

The proposed scheme relies heavily on the extensive use of encryption and hash codes in order to ensure secure data access through secure entities and secure communication. The scheme maintains various files for different entities in the system. Among the entities of interest in the scheme both administrators and users require their details to be

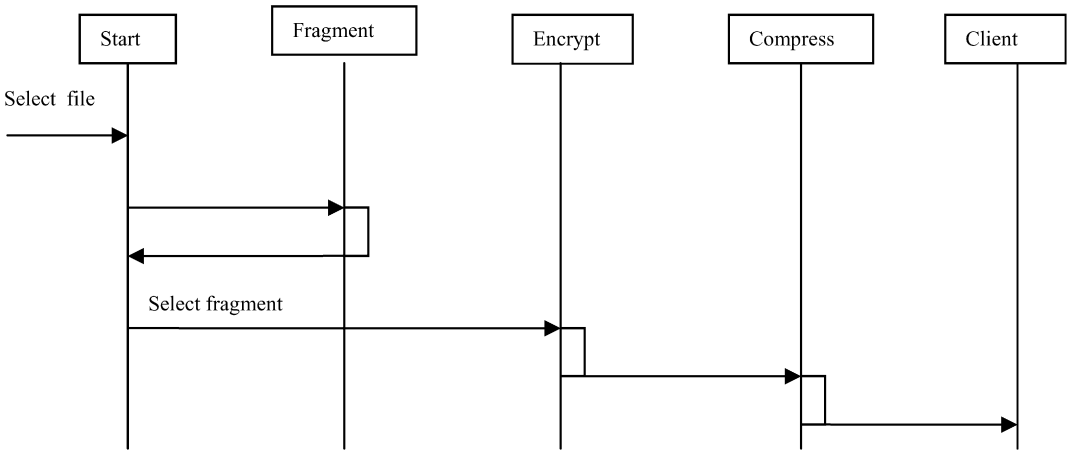

Fig. 3. Component interaction.

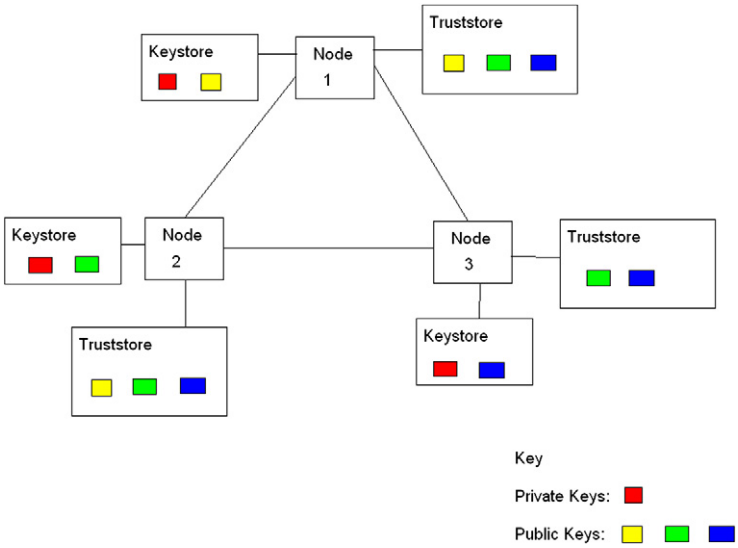

Fig. 4. Secure communication.

held in secure form. As stated earlier, the list of administrators and the list of the users are stored by converting their usernames and passwords to a hash code using the Secure Hash Algorithm (SHA).

Another entity of importance is the data fragment. Each cipher text fragment held in this directory is given a unique identification before it is transmitted and stored. Each cipher text fragment ID is calculated as follows: a hash value is calculated for the original plaintext datum using the Secure Hash Algorithm (SHA); this hash value is then concatenated with the time the cipher text fragment was created and a random number; this concatenated byte code represents the ID for the cipher text datum.

3.3. Secure communication

This module provides a means for secure communication through the use of SSL between a client and a set of servers. This module contains a key store and a trust store. Each node has its own public, private key pair and the trust store contains every public key i.e. one for each node in the system, held within an X.509 certificate. The key store contains the public, private key pair for the local machine. Any connection to the server from any outside process requires authentication through SSL. The client must have an X.509 certificate, which is held in the trust store of the server it is trying to communicate with, indicating that the client can be trusted.

The diagram in Fig. 4 depicts a small network of nodes and is used to describe the use of SSL in securing communication channels, and preventing unauthorised connections to the server process running on a node. In the diagram each node has a corresponding key store and trust store held in the communication module. The key store of each node contains a public and private key pair for that particular node, whilst the trust store of each node contains the public keys (held within X.509 certificates), which that node trusts. In the diagram *Node* 1 and *Node* 2 contain the public keys of all nodes in the system. As a result the server process running on each of these machines will accept incoming connections from any client process held on any of the three nodes. However since *Node* 3 contains only the public keys for itself and for *Node* 2, it follows that only the client processes on *Node* 2 and *Node* 3 can communicate with the server process running on *Node* 3. The public key for *Node* 1 is not held in the trust store of *Node* 3; as a result the server running on *Node* 3 will not authenticate the client attempting to make a connection on *Node* 1.

3.4. Reliability

The recoverability of file fragments from the servers depends on the reliability of the underlying system. In the current implementation, the scheme relies on the host operating system and the network for the provision of a quality of service. This approach is clearly inadequate and the recoverability of state should be an integral part of the scheme. This can be achieved by regular back up of data to external media by the server processes, a matter that is being investigated for further work.

4. Performance

Some of the strengths and limitations of the scheme can be identified by observing its behaviour under specific operational conditions and monitoring its performance. Although the encryption of the files is critical to the scheme, it is a feature that is common to most cryptographic systems. The file fragmentation and distribution are, however, distinctive features of the scheme and their performance under different modes will put the scheme in an adequate context. Three types of measurements were generated, by varying the size of the files and the number of nodes:

(1) Uploading and downloading files of various sizes and monitoring the time taken to complete each operation. The performance is shown in Fig. 5.
(2) The effect of data compression on the utilisation of disk space for files of differing sizes (Fig. 6).
(3) The impact of an increase in the number of nodes in a system on the time taken to upload and download a 100 K file (Fig. 7).

The results show that it is not practical to process files that are much larger than a megabyte due to the large amount of time taken by file processing. This could be overcome by enabling the user to choose the size of the plain text fragment (it is 1 K in this implementation). If the user were able to increase the size of the plain text fragment the

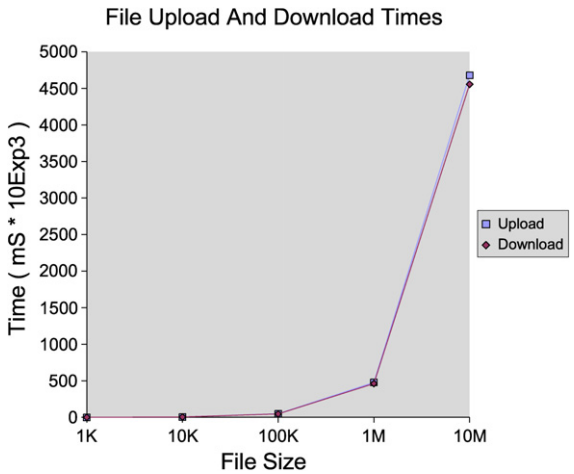

Fig. 5. Upload/download times.

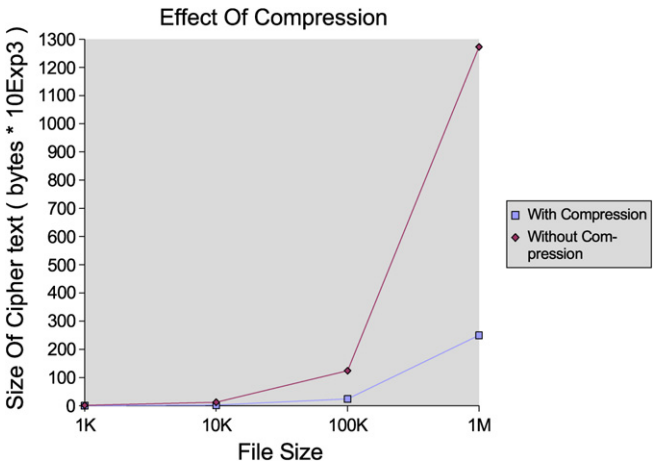

Fig. 6. Effect of compression.

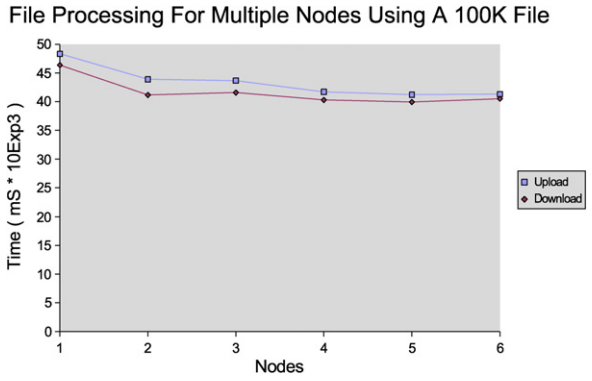

Fig. 7. Network effect.

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6
Node 1	21	21	25	27	22	19
Node 2	26	31	33	26	25	30
Node 3	22	25	18	21	29	25
Node 4	31	23	24	26	24	26

Fig. 8. Fragment distribution.

file would be split into fewer fragments. Theoretically, with coarser granularity file processing should be faster since there would be fewer symmetric keys to generate and less switching between encryption algorithms. This measure may increase the response times for large files, but may lower the level of the security offered by the system, since the degree of distribution of the file is reduced. The quality of service offered by the scheme depends to a large extent on the network bandwidth, the availability of memory and the load average. If the network bandwidth and the amount of memory (whether persistent or volatile) is low and the load average is high then the performance of the scheme is bound to decrease. It is also worth noting that the introduction of file compression has a significant effect on performance. Its effect is to bring the size of the file to an acceptable level that can be processed effectively by the system. Another aspect of the behaviour of the system is that the increase in the number of nodes beyond two does not improve the performance of the scheme significantly. The overheads associated with the fragmentation and the defragmentation of files form a significant part of the performance.

The performance of the system was also investigated in order to determine the effect of the random allocation of the fragments to the four nodes. In Fig. 8 the table shows the distribution of 100 fragments (100 K file) across 6 runs. An equal distribution would have resulted in 25 fragments in each node. With random distribution, however, the table and its graphical representation in Fig. 9 show that a deviation of 5 fragments from a standard value does not lead to an imbalance in the load between nodes. The random nature of the distribution has the added advantage that it prevents the generation and the detection of patterns of behaviour in the file system.

5. Related work

The proposed scheme represents one point in the spectrum of distributed file system designs for secure data access. It combines encryption with file fragmentation and distribution in order to achieve both file data concealment and file location concealment. These characteristics are associated with a number of factors: the granularity of the file encryption, the key and metadata management, the level of concealment of the data and the file location, the degree of distribution and fragmentation of a file, the ease of access to original data, and the level of implementation, namely user level or kernel level.

Most of the proposed systems for securing data operate either at directory or at file level, and conform largely to the model introduced in CFS [4]. There is a notable exception to this mode of operation and its coarse granularity. Banachowski et al. [13] present a system for intra-file security. It allows for the selective encryption of blocks of data within the same file. It is akin to the proposed system in its granularity, since it operates at block level and therefore at

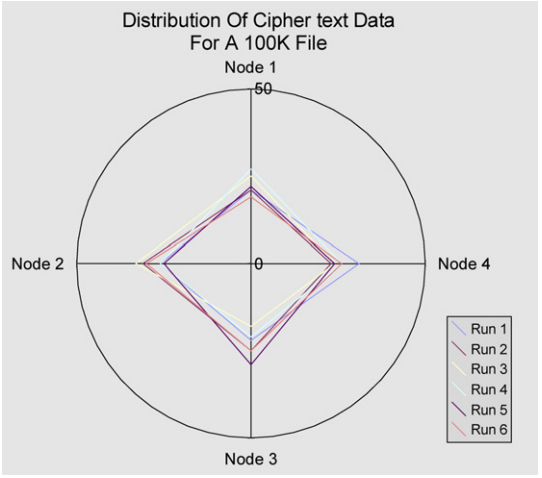

Fig. 9. Cipher text distribution.

byte level. The fragments are, however, encrypted *in situ* and kept on the local machine. This approach offers greater flexibility and availability, but requires additional metadata information and key management services. It does not offer total file location concealment, however. The file is still visible and can be tampered with locally. This scheme is inappropriate for applications that require total concealment of data. A typical application for this model of encryption is the scientific domain where sensitive data can be encrypted and the rest of data left in plain text.

Key and metadata management are obviously crucial to the function of cryptographic systems and their potential application is the subject of intense research and debate. The choice of symmetric or asymmetric encryption, the role of login passwords for encryption, the way keys are stored can determine the level of security of the schemes as well as their performance [14]. As for the management of metadata, the issue revolves around what information to generate, where to store this information, namely on-line or off-line, and under what form, plain text or cipher text. All these features define a range of models that can offer secure access to data. In some systems the metadata is fully encrypted [15], while in others it is partially encrypted in order to allow normal administrative procedures to take place [4]. In Banachowski's scheme, for example, the availability of the whole file locally and access to its metadata offer greater flexibility [13]. Blaze describes a system where keys are kept off-line on a smartcard [5]. This scheme is an attempt at exploiting the fact that file encryption keys are distributed temporally and have a longer lifetime than the keys used for communication. These systems maintain the full structure of the file locally. They are also characterised by the fact that they are concerned with key management only [16]. Metadata is dealt with in a traditional manner and is deemed to be the responsibility of the host operating system. Figure 10 presents a comparison of four schemes in terms of the criteria presented earlier.

The scheme proposed in this paper contrasts very strongly with traditional architectures, and addresses many of the limitations of classical file systems. It circumvents their centralised features through a symmetric client server model, and operates at a higher level. This scheme can be incorporated with Web browsers and applications by adding an application protocol layer. Security and confidentiality are dealt with by means of symmetric schemes

	Granularity	Key management	Metadata management	Level of concealment	Degree of distribution	Level of implementation
CFS [5]	Coarse (file)	Yes (on-line)	Yes (on-line)	Total (data, metadata)	Low (NFS based file)	User
TCFS [7]	Coarse (file)	No (OS management)	No (OS management)	Partial (data)	Low (OS based file)	Kernel
Banachowski scheme [13]	Fine (byte)	Yes (multiple pairs)	Yes (flexible)	Partial (in situ)	Low (OS based file)	User
Proposed scheme	Fine (byte)	No (off-line)	No (off-line)	Total (data, file location; off-line)	High (fragments across servers)	User

Fig. 10. Scheme comparison.

while authentication is addressed by the adoption of asymmetric schemes as inherent components in client server interaction. The scheme proposed in this paper is eclectic in many ways. It combines various features, such as finer granularity of encryption, since it operates at block/byte level, and stores not only the keys but also the metadata on a physical medium, off-line. In addition to this non-availability of metadata, it achieves total file concealment through the distribution of encrypted and compressed fragments. Of particular importance in the scheme is the fact that encryption/decryption keys and metadata are kept off-line on a physical medium such as a disk or smart card. There is therefore no specific requirement for the management of keys or metadata since they are removed from the system.

6. Further work

One distinguishing feature of this scheme is that secrecy is achieved at the expense of availability since metadata is kept off-line. Furthermore, loss or damage of the medium can lead to denial of service. This approach requires effective security procedures to be applied to the medium. Although the system offers transparency with respect to the operating system and is portable, it suffers from a lack of flexibility in retrieving the original file. The scheme is however relatively easy to use, and is geared towards application areas where sharing of data is not vital, and where secrecy and confidentiality are paramount. Discrete parts of the file can be encrypted selectively, a facility that subsumes the scheme presented in [13]. This scheme is implemented as a prototype, and is still under development. A number of issues and improvements are being considered:

- Automation of key management, in particular, the addition of a X.509 certificate (used for process authentication) throughout the system, to automate the removal of a X.509 certificate (used for process authentication) from the entire system [17]. A balance needs to be stuck between the role of public keys in order to satisfy the requirements for secure communication, and the use of symmetric keys in the encryption and decryption of the file fragments.
- Variation in the granularity of the text by allowing the user to select the size of the plaint text fragment before file processing. This facility will make fragments suitable for load balancing [18]. The configurations of the nodes may be different, thus warranting a more judicious allocation of fragments to nodes with higher capacity than others.
- Improvement to the robustness of the scheme by providing a means of system recoverability should any node on the system fail. Other than through the regular backup of cipher text data held on a machine/node, there is currently no other means of system recovery should one or more nodes crash.
- Investigation of the performance of different encryption algorithms in order to reduce the overheads of the encryption/decryption process [19]. The use of a variety of algorithms in conjunction with variable length fragments will reduce the occurrence and manifestation of patterns. This investigation should also consider scalability issues that arise from an increase in the number of nodes.
- Provision of a more efficient and selective read/write mechanism. This can be achieved by calculating a hash value for each plain text fragment, and by storing it with all the other metadata in the file table. Instead of deleting the entire file held on the distributed system, only the plaintext fragments whose new calculated hash values differ from the previous hash values, held in the old file table, are encrypted and distributed across the system. The hash value can also be used to enforce integrity constraints [4].
- Enhancement to the system in order to improve transparency with respect to users, through the automation of some of the steps involved in the data file concealment procedure. The proposed automation should not, however, compromise the stateless characteristics of the system. Implementing an application protocol layer in order to facilitate its incorporation with Web-based applications can also enhance the level of transparency.

7. Conclusion

The proposed scheme is aimed at enhancing secure access to data within a distributed system. In addition to file data concealment, it offers transparent file location concealment, through a combination of distribution, fine data granularity and encryption. Files are broken up into segments, which are encrypted and then distributed randomly across a distributed system, with the metadata stored off-line. The scheme is implemented at user level and promotes a viable approach to the secure storage and access to files in distributed systems, such as the Web. Although this

high level of secrecy is achieved at the expense of flexibility and performance, the system is portable across different platforms and offers transparency with respect to operating system and network.

References

[1] E. Miller, W. Freeman, D. Long, B. Reed, Strong security for network-attached storage, in: Proceedings of the FAST 2002 Conference on File and Storage Technologies, Monterey, California, USA, 2002, pp. 1–13.
[2] B. Schneier, Applied Cryptography, second ed., John Wiley & Sons, 1996.
[3] S.T. Vinter, Extended discretionary access controls, in: 1988 IEEE Symposium on Security and Privacy, Oakland, USA, April 1988, pp. 39–49.
[4] L. Ferreira, A. Zuquete, A. Ferreira, SEFS: Security module for extensible file system architectures, in: European Conference on Information Security Solutions, ISSE, Barcelona, September 2000.
[5] M. Blaze, Key Management in an encryption file system, in: Proceedings of the First ACM Conference on Computer and Communication Security, Fairfax, VA, USA, 1993, pp. 9–15.
[6] R. Sandberg, D. Goldberg, K.S.D. Walsh, B. Lyon, Design and implementation of the sun network file system, in: Proceedings of the Summer 1985 USENIX Conference, Oregon, USA, June 1985, pp. 119–130.
[7] G. Cattaneo, L. Cattuogno, A. Del Sorbo, P. Persiano, The design and implementation of a transparent cryptographic file system for UNIX, in: Proceedings of the Annual USENIX Technical Conference, FREENIX Track, 2001, pp. 245–252.
[8] B.C. Reed, E.G. Chron, R.C. Burns, Authenticating network-attached storage, IEEE Micro (January–February 2000) 2–10.
[9] A. Harrington, C. Jensen, Cryptographic access control in a distributed file system, in: SACMAT '03, Como, Italy, June 2003, pp. 158–165.
[10] D. Mazieres, D. Shasha, Don't trust your file server, in: Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems, HotOS-VIII, May 2001, pp. 99–104.
[11] A. Spalka, H. Langweg, Notes on application-oriented access control, in: Proceedings of the 13th International Workshop on Database and Expert Systems Applications, DEXA Workshops, 2002, pp. 451–455.
[12] E. Levy, A. Silberschatz, Distributed file systems: Concepts and examples, ACM Comput. Surveys 22 (4) (December 1990) 321–374.
[13] S.A. Banachowski, Z.N.J. Peterson, E.L. Miller, S.A. Brandt, Intra-file security for a distributed file system, in: Proceedings of the 10th Godard Conference on Mass Storage Systems and Technologies, College Park, MD, 2002, pp. 153–163.
[14] C.P. Wright, J. Dave, E. Zadok, Cryptographic file systems performance: What you don't know can hurt you, in: Proceedings of the 2003 IEEE Security in Storage Workshop, SISW, 2003, pp. 47–61.
[15] E. Riedel, M. Kallahalla, R. Swaminathan, A framework for evaluating storage system security, in: Proceedings of the FAST 2002 Conference on File and Storage Technologies, Monterey, California, USA, 2002, pp. 28–30.
[16] D. Mazieres, M. Kaminsky, M.F. Kaashoek, E. Witchel, Separating key management from file system security, in: Proceedings of the 17th ACM Symposium on Operating Systems Principles, SOSP '99, Kiawah Island, SC, December 1999, pp. 124–139.
[17] E.L. Miller, D.D.E. Long, W.E. Freeman, B.C. Reed, Strong security for network-attached storage, in: Proceedings of the FAST 2002 Conference on File and Storage Technologies, Monterey, USA, January 2002, pp. 1–13.
[18] R. Anane, R. Anthony, Implementation of a proactive load sharing scheme, in: ACM Symposium on Applied Computing, SAC, Melbourne, Florida, USA, 2003, pp. 1038–1045.
[19] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyarayanan, R.N. Sidebotham, M.S. West, Scale performance in a distributed file system, ACM Trans. Comput. Systems 6 (1) (1988) 51–81.