SCFS: TOWARDS DESIGN AND IMPLEMENTATION OF A SECURE DISTRIBUTED FILESYSTEM

Juan Vera-del-Campo, Juan Hernández-Serrano and Josep Pegueroles
Politechnic University of Catalonia, Jordi Girona 1-3, Barcelona, Spain

Keywords: Peer-to-peer, Distributed Filesystem, Security, CFS.

Abstract: Our digital world creates lots of data than users desire to preserve from malfunctioning, local disasters or human errors. Current nodes in the internet has enough intelligence and processing power to allow the deployment of distributed services on common nodes. This is the case of peer-to-peer networks and services. There are several proposals in literature to deploy a distributed filesystem over the internet. This paper presents and analyses the security of a prototype based on Cooperative File System.

1 INTRODUCTION

Data is for many people one of the most valuable possesions. Very often, personal data is stored in hard disks or removable devices that are both fragile and localised. Furthermore, there are many situations that lead to loss of data in this kind of media, such as human or computer errors, fire or water damage accidents or stealing of hardware. Data is even a more important resource for enterprises, that are willing to pay for expensive and complex mechanisms to ensure its availability.

Distributed file system spread data in many nodes of the internet in order to protect it against local malfunctioning hardware. Today most nodes in the internet are driven by common people and they have not special roles in the network. There is plenty of bandwidth and digital intelligence that is wasted in personal computers around the world. Peer-to-peer networks use the latent power and bandwidth of common nodes in the internet to provide decentralised, distributed services that seem very convenient to provide reliability to a distributed filesystem without assuming the cost of high-end internet servers.

But distribution of data has a main drawback regarding security. Many people will object if their data is stored in the computer of a complete stranger, since he will be able to access, for example, to their sensitive bank accounting information. Even collecting the name of the files that a user has in his personal namespace is a violation of his privacy. A system that focuses on providing reliable storage of personal data and backups must face security as one of the main objectives of design.

In this paper we propose a secure architecture for the storage of personal data in distributed peer-to-peer networks. The main objective of the proposal is to provide a reliable service that clients will use to backup their valuable and sensitive personal data, without fear of data losing or undesired spying.

The structure of this paper is as follows. Section 2 studies several distributed file systems that focus on security and their weaknesses. Section 3 analyses CFS, presents the main scenario of application and explores the requirements of the service. Section 4 presents our proposals to secure CFS. Section 4.8 analyses the security of our prototype implementation. Finally, the paper ends with the conclusions and references of our work.

2 RELATED WORK

There are many proposals on distributed file system in literature. From NFS to AFS, many widely deployed distributed file systems rely on several more replicated servers. But this approach is neither scalable nor cheap, and servers are a honeypot for attackers, even if distributed. To solve these problems, distributed filesystem over peer-to-peer appeared in literature. In this section we will summarise some of the properties of distributed and decentralised filesystems that has a strong focus on security. Readers can consult a general analysis of distributed filesystems

Vera-del-Campo J., Hernández-Serrano J. and Pegueroles J. (2008).
SCFS: TOWARDS DESIGN AND IMPLEMENTATION OF A SECURE DISTRIBUTED FILESYSTEM.
In *Proceedings of the International Conference on Security and Cryptography*, pages 169-176
DOI: 10.5220/0001923901690176

in (Hasan et al., 2005).

MojoNation (MOJ, 2000) was a robust, decentralised file storage. Nodes were organised in a peer-to-peer network with ring shape. Files were broken down in blocks that were replicated and stored in the network using a unique identifier per file. The company that deployed MojoNation commercialised cash units, called mojos, as its bussines model. Mojos were used to prevent abuse and flooding. MojoNation died because original designers didn't consider the high rate of joins and leaves of common nodes.

Free Haven (Dingledine et al., 2001) uses a non-structured peer-to-peer network and routes searching messages by flooding the network. Files are broken down in N blocks using an information dispersal algorithm (Rabin, 1989) Clients create a pair of public-private keys for each file. Each block is indexed with the same key, $hash(PK_{sub})$, and traded with a closed list of neighbours taking into account reputation. When asking for a key, clients wait for $k < N$ parts and reconstruct the whole file. Nodes trade blocks with other nodes to improve anonymity and persistence of data.

FreeNet (Clarke et al., 2000) was designed to store common data and to allow its later access by means of an associated key, preventing the censorship and offering anonymity to the user who publishes the document and to the one that downloads it. To achieve these goals, the Freenet network creates a not-hierarchical and not-structured organisation of nodes that anonymously transmit and cache messages and documents among them.

GnuNet (Tatara et al., 2005) has the objective of building a broadcast routing algorithm based on specific nodes currencies of nodes. Each node valuates its neighbours based on their behaviour and the number of messages that route or ask to route. Messages from less valued nodes receive less priority in the output queue. Each node routes its messages based on its own interests, but the economics of the network supports strong collaboration.

Over this routing algorithm a file sharing protocol was developed (Grothoff et al., 2006). Files are divided in blocks B_i that are individually encrypted using $H_i = hash(B_i)$ as key, and identified by $H_{ii} = hash(H_i)$ in the network. Nodes trade these blocks with neighbours, and get other blocks in exchange. Users searching for files send the hash of the keyword to get the data blocks.

Shark (Annapureddy et al., 2005) divides files in blocks and uses a DHT to store the address of nodes with of them (proxies). When a node downloads a chunks, it publishes itself as holder of a replica. Only nodes that are controlled by the same user will be proxies of personal data, so clients need at least one of their nodes to be connected to retrieve data. Clients must prove knowledge of the contents of a chunk to read/write, and then anonymity is not achieved.

Cooperative File System (CFS (Dabek et al., 2001)) is a file system over a distributed hash table. Since this filesystem is the core of our proposal, it will be explained in detail in the next section.

3 SCENARIO OF APPLICATION

The scenario of our study is as follows. Individuals want to use the internet as a backup of their personal files, or even as a nearly unlimited disk space for little devices. Distribution of their personal data is not their main objective, but preservation of their own and unique data and accessibility from any device that they may own. In addition, they are really interested in keeping their data private and out of the reach of both casual and commercial eyes.

The features of structured Peer-to-peer networks are very convenient to achieve reliability and accessibility of personal data to thousands of users at the same time. Among the networks that were studied in section 2, CFS is the only structured Peer-to-peer network. Authors of CFS didn't consider security in their original design, and this paper describes the construction of a secure structure over CFS. We call this system Secure Cooperative File System (SCFS).

3.1 Cooperative File System

Our proposal aims to add security to the Cooperative File System (CFS (Dabek et al., 2001)), that is a file system over a Distributed Hash Table (DHT). Figure 1 shows the architecture of this algorithm.

A DHT is a structured peer-to-peer network where nodes pick up a random identifier ID_i. Each one finds and links at least to the node that has the very next $ID_n > ID_i$ in increasing order. The node ID_i is in charge of storing data that is identified with an $ID \in (ID_i, ID_n)$. The process goes on until that the last node links to the first one and the whole network creates a ring structure. In order to put or get the information identified with ID, a node sends clockwise a message in the ring to the node in charge of ID, that answers. To improve the routing in the ring, nodes have far links to other nodes of their choice. Current implementation of DHTs include Kademlia (Maymounkov and Mazières, 2002) or Chord (Stoica et al., 2001), and the main difference between them is the specific algorithm for far links.

CFS was implemented over Chord. It gets a file F with a filename f, divides the file in blocks, stores the blocks B_i and calculates H_i and ID_f.

$$F = \{B_1 \cup B_2 \cup B_3 ... \cup B_n\} \quad (1)$$
$$H_i = hash(B_i) \quad (2)$$
$$F_h = \{H_1, H_2, H_3, ..., H_i\} \quad (3)$$
$$ID_f = hash(f) \quad (4)$$

Then CFS saves each block B_i in the DHT identified as H_i and stores the list of H_i in a special block under ID_f. Files in CFS are persistent and the system warrants that every file will be retrieved, regardless of its popularity.

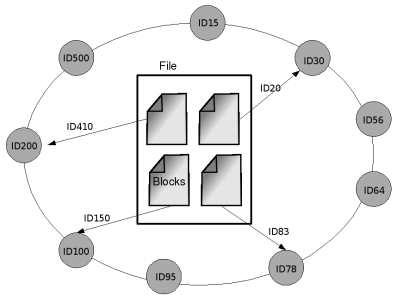

Figure 1: Cooperative File System.

3.2 Security Requirements

Distribution of personal data in peer-to-peer networks has many drawbacks from the point of view of security. Many people do not want that strangers were able to access to their private data, and even they will object if it is possible to get the knowledge of the existence of some files. Intruders may look for common file names as “accounting”, “strategic plan” or “passwords” in the whole network for any legitimate user. In addition, in some countries having a single MP3 file or some kind of adult content may be severally prosecuted.

The security requirements of SCFS are:

Confidentiality. Data should not be readable for others apart from the user than uploaded it. In a distributed filesystem every node stores personal data from any user, and the filesystem must supply mechanisms to warrant that node administrators cannot read data from other users. On the other hand, if nodes cannot read the content that they store they can positively deny its knowledge and protect themselves from legal prosecution for storing and distributing illicit content.

Privacy. Malicious users may collect data about habits or interests of users. Even if an attacker is not able to access to actual data, the name of the files of the user namespace may be relevant. Commercial research and dictatorial governments may get profit from capturing the names of files accessed by users. The filesystem must provide methods to prevent such eavesdropping.

Integrity. Since data is stored in uncontrolled nodes, it is not possible to prevent the modification of data. The filesystem must provide mechanisms to detect modification and restore original data, if possible.

Persistence. The filesystem must prevent data losing. Files can be lost by means of malicious nodes that remove pieces of data, users that write data in the same place, both intentionally and unintentionally, or network or node failures. Storage systems focus on persistence instead than on publishing of data.

Availability. Users are in the move and own many devices with different network capabilities, memory and processing power. In spite of this, they want to access to a consistent disk space from every device regardless of its capabilities. The system should provide mechanisms to allow data to be available from any of the devices of the users, regardless how and where they are connected.

In the next section we propose several mechanisms on top of CFS to cover these design objectives.

4 SECURE COOPERATIVE FILE SYSTEM

Distributed file systems over structured peer-to-peer networks provide the requirements that the scenario under study needs. CFS is the first choice to develop a distributed file system for personal files, but it does not provide the necessary security services. In this section we propose many mechanisms to add to the standard CFS system in order to cover the security objectives studied in section 3.2.

This section will explain in depth each of the steps. A graphical outline of the whole process is shown in figure 2.

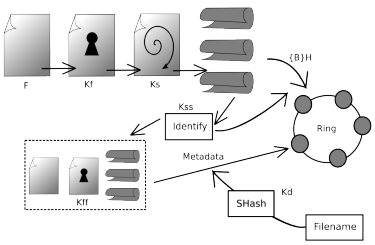

Figure 2: Description of the process.

4.1 Assumptions and Definitions

Table 1 includes the definition of the main concepts of SCFS. Readers will find a relation of symbols in table 2.

Table 1: Definitions used in this paper.

User	A human client of the system.
Node	Each one of the devices that a client uses to join to the network.
File	An ordered array of data
Directory	An unordered set of files and directories
Filename	A human readable identifier for a file or directory. It is not unique in the system, since two different users can store different files under the same filename
Root Directory	The filename of the directory that holds every file and directory of the user. Altough not mandatory, we assume that users will have a root directory for all his documents and files.
Block	Each one of the pieces than the IDA creates from a file
Metadata Blocks	Special blocks of data that have enough information to restore the complete file
iNode	The first block of metadata that holds a list of the rest of metadata blocks

In order to gather all this information in a single point, configuration file exists. A configuration file may be associated to a file, a directory and its contents or every single document of the user. Configuration file store the identifier of the root directory of the user, if needed, and the set of keys K required to get the associated object or set of objects. These configuration files are stored locally to the user in each one of the nodes and kept private.

For example, Bob and Alice join to a SCFS network. They both desire to store a file that has "revenues.xls" as filename. Bob has $UID_{bob} = bob$ as his identifier, and Alice $UID_{alice} = alice$. Bob has $cfs://bob/root/$ as the root directory, and Alice has $cfs://alice/root/$. In order to access to the personal root directory of Bob, he will need his global key K_d. Since their user identifiers and K_d are different, their namespaces will be different as well, as the next section shows.

4.2 Securing the Whole File

The first step to secure a file is encrypting its contents. Each file is encrypted with a symmetric algorithm with K_f. Since there are additional steps that enhance the security of the system and an attacker cannot get the whole file from one or several blocks, an encrypter as strong and slow as AES is not really needed. Weaker but quicker algorithms such as RC4 are desirable for little devices. Encrypting a file ensures that casual attackers won't be able to sniff its contents.

After the encryption process, a shuffling and redundancy creator algorithm takes place. SCFS uses an information dispersion algorithm (IDA) described in (Rabin, 1989). This step has a double objective. The first one is preventing that consecutive bytes in the original file were consecutive in the stored blocks. This shuffling prevents some kind of statistical attacks against files with a known structure or common header, such as PDF files. Since users can choose the kernel space of vectors for the IDA algorithm, the spreading of data in the final blocks is deterministic only for the original author. Key K_s is used to generate the vector space that the IDA algorithms need. The second objective of the algorithm is creating redundancy of data. In order to enhance the availability of the service, the algorithm takes the original file and created m blocks in such a way that it will need $k < m$ blocks to restore the original file. During the reading process that redundant information may be used to check the integrity of data and correct errors. In this sense, the system is protected against malicious nodes that randomly changes data that they store, overloaded nodes than cannot manage all their parallel connections and local network failures. The dimension of B_i is chosen in such a way that it matches the size of a block in the DHT.

$$S = \{F\}_{K_f} \tag{5}$$

$$B = \{S\}_{IDA} = \{B_1, B_2, \ldots, B_i\} \tag{6}$$

$$dim(F) = dim(S) < dim(\cup B) \tag{7}$$

Clearly, this algorithm consumes both time, bandwidth and disk space in remote nodes since the dimension of the final data in B is greater than the dimension of the original file F. Users can choose the amount of redundancy to apply to each file, or even if this algorithm runs or not at all over their files. Authors expect that clients will use most of the time the default redundancy of 30%.

The encryption and shuffling steps take place locally in a node and the set B is not yet published in the ring.

4.3 Securing File Blocks

After the previous step each block B_i has the same size than the DHT blocks. Since the data in each B_i was encrypted and shuffled, it makes little sense to any casual attacker that sniffs the blocks.

Table 2: Symbols used in this paper.

UID	The identifier of each user. UID is the same for every node that a user controls.
F	The original file of the user
ID_f	The final identifier of a file
S	The encrypted file of the user
B_i	Each one of the blocks of a file
B	The set of blocks of the file
H	The set of identifiers of blocks
K_d	The key used to secure the filename
K_f	The key used to encrypt the file
K_{ff}	The key used to encrypt the metadata
K_s	The key used to generate a vector space for the IDA algorithm
K_{ss}	The key used to create block identifiers
K	The set of keys. It can be generated from a master key associated to a single file.

The CFS stores and retrieves blocks of data associated to a identifier. We propose three different methods to identify each one of the blocks. All of them have their advantages and disadvantages.

Random Identifiers. The local node picks up a random identifier for each block. The identifier has the same size than identifiers in DHT. This is the more secure method since the random identifier has not information about the block, its contents or its publisher. On the other hand, the set of ordered random identifiers of the file blocks must be stored by the user locally.

$$H = \{random_i\} \mid dim(H) = dim(B) \tag{8}$$

A file of 20MB that is stored in blocks of 2048B need about 160KB to store the random identifiers. Since SCFS uses special blocks to store file identifiers, this approach needs at least 80 blocks just to store the file structure.

Pseudorandom Identifiers. Identifiers for each block are created with a pseudorandom algorithm using K_ss as seed of the algorithm.

$$H = \{K_{ss}\} \tag{9}$$

In this case the user only have to store a key of 128 bits for every file in the system, no matter of its length. K_{ss} must be kept private, since there is no need to make the job of an attacker easier publicising the list of blocks.

Hash of the Block, both basic and secure hash. With the same considerations as random identifiers in respect to size, hashes have the advantage that they could be used to ensure integrity of data or event prevent unauthorised overwriting, as will be stated in next sections. These two advantages are enough to make advisable to calculate the hash code of each block and store it in the metadata file.

$$H = \{hash_{K_{ss}}(B_1), \ldots, hash_{K_{ss}}(B_i)\} \tag{10}$$

4.4 Metadata

In this moment the user's node has a local buffer with the blocks of the file B, a set H to identify each block and a configuration file, as stayed in section 4.1. Metadata is then managed in the same way that a file object. It is padded, encrypted with K_{ff} and divided in blocks using IDA. Many of the special data is stored in a special block that we call iNode, in the same sense that iNodes in the traditional filesystems. iNodes store the identification of the set K, the whole contents of H and the identifiers of the rest of metadata blocks.

In SCFS metadata blocks are published as regular blocks. Since they have the same length and entropy as any other block, they cannot be distinguished from regular blocks. Identifiers are assigned in the same way as block identifiers. The iNode is identified in a special way, described in the next section. The first block is referred with a special identifier. Metadata blocks are then randomly introduced in the local buffer. In this way an eavesdropper cannot discriminate between file blocks and metadata blocks.

4.4.1 Securing File Identifications

Every file in SCFS is related to a URI in a injecting relation. This means that an URI identify a file and any file is identified just with one URI. Furthermore, it should not be possible to find out the URI from any number of parts or blocks of the file.

Users store in SCFS their private data with an arbitrary name, and the file system must provide with a separate namespace of files for each one of the users. In this sense, a human readable filename must be the main interface of the user to the system. The complexity and security involved in mapping that string to a DHT identifier must be hidden to the user.

If an attacker is able to identify the iNode, with convenient cryptanalysis he may be able to retrieve the complete metadata and then the complete file. In this sense, securing the identification of the first metadata block is crucial for the security of the system.

There is a private namespace for each user in SCFS. Private namespaces have an associated secret key K_d. The filename is hashed and then encrypted using this key. The hashing step maximises the entropy of the encrypted string. The encryption step warrants that the userspace is unique and secured,

since none can identify the file even if he knows the author and the filename.

$$ID_f = hash_{K_d}(UID, filename) \quad (11)$$

4.5 Publishing the File

At this moment, the node has a local buffer with a set of blocks B that are identified with H and the iNode of the system identified with ID_f. The node publishes the contents of the local buffer in the DHT under they keys. Readers will notice that since blocks, metadata and the iNode were randomly introduce in the local buffer, an attacker is not able to put them aside and a cryptanalysis is really difficult.

4.6 Reading Process

The reading process is the inversion of the writing process. From a filename users create a ID_f, maybe using a user key K_d to maintain privacy. From ID_f, the user gets the iNode of the file and then the list of metadata block. From the metadata, the user is able to recreate H. Then, he downloads the blocks, deshuffles and then decrypts the file.

4.7 Implementation and Additional Mechanisms

The ideas in the previous sections were implemented in a prototype accessible in (del Campo et al., 2008a). The prototype uses Kademlia (Maymounkov and Mazières, 2002) as the algorithm for the DHT, blocks have 2048 bytes and there is a 30% of redundancy in the IDA algorithm. Identifiers of files, nodes and users have 128 bits. AES is used as the encryption algorithm, and the first 128 bits of SHA as a hashing algorithm. Kademlia was chosen as the DHT algorithm since its buckets are more stable against ring breakages than Chord, the DHT that CFS uses. The prototype is written in Python and released under the GPL license.

Apart from the mechanisms of the previous sections, our implementation of SCFS includes others to enhance the security of the system. In this section we will study these additional mechanisms that take place in phases other that reading and writing data.

Attackers may join a large number malicious nodes in order to perform a Sybil attack. In this way, the attacker gains a large influence in a region of the Kademlia ring and he may be able to put the complete system down or perform denial of service attacks to some users. In order to prevent this kind of attack, the original designers of CFS proposed that the identifiers of the nodes must depend on the network address of the node. Furthermore, the random identifiers of each block disperse blocks in the whole ring, while the IDA warrants that the file can be retrieved even if a segment of the ring is not available. Kademlia is especially strong against breakage of the ring (Maymounkov and Mazières, 2002).

Attackers may collect information by means of sniffing the communications of users in the network. Files are encrypted, shuffled and splitted down in blocks as explained in section 4.3, but metadata have a slightly weaker encoding process, specially the first of them. An attacker that gets the first iNode needs to decrypt a single block of data to get the list of every blocks in the system. That first iNode is encrypted using AES, but it contents known information that may simplify the cryptanalysis. An attacker may identify this first inode because it will be the first block than a user demands. In this way, SCFS asks for several random blocks apart from the iNode.

There is no deletion service in SCFS. A deletion service needs to authenticate the owner of the file in order to prevent deletion from unauthorised users. In order to enhance privacy of the users, SCFS does not include any authentication mechanisms. In order to prevent exponential growing with time, data is actually deleted in nodes if owner does not access to the block after a month. In order to show interest for a block without the need of downloading it and increasing the bandwidth, users must send a “ping” to the data block. Nodes count these pings as an access and will mark the block as no removable for an extra month.

Both malicious and fair users may overwrite blocks of legitimate users. Since identifiers of blocks are random numbers, collisions are possible. Kademlia supports storing different blocks under the same key, and when a user demands that key he will receive the whole collection. SCFS takes advantage of this by means of saving the hash of the block in the iNodes. In this sense, when a user demands a key and receives several blocks, he is able to discriminate which one is the valid to create the original file. This mechanism does not prevent that malicious users are able to write blocks under the same key, but their blocks won’t be used to recreate the original file.

Directories are files with an unordered list of file or directory identifiers than it holds. Raw content of a directory is managed as any other regular file in the system. SCFS uses different URIs to distinguish between files and directories.

4.8 Security Analysis

In this section we study some possible attacks against the system and their counter measures. These attacks are based on the objectives that were listed in 3.2. For a detailed comparison of SCFS with other distributed file systems, the reader can consult (del Campo et al., 2008b).

An attacker eavesdrop user's communications. Every piece of data is locally encrypted and it is never stored in plain text on SCFS. The objective of this attack is getting the root directory, the iNode or a collection of blocks of a file to perform the next attacks. SCFS asks for a number of random blocks in the first place. File blocks and metadata are indistinguishable and all of them have the same size. SCFS makes difficult for an attacker to put file blocks apart from metadata or directories.

An attacker wants to get a particular file of a user. The attacker knows of the existence of a file named in a certain way in the personal namespace of the user, as "income.xls". SCFS uses the secure hash of a filename to identify an isolated file. In this case, the attacker must break down the secure hash to get the iNode, and decrypt this block to get the list of blocks of the file. Our prototype uses random identifiers for files stored in a directory, and only the name of the directory of the root directory uses secure hash with K_d. Random identifiers are safe against this attack.

An attacker wants to list the contents of a directory Directory names are protected with K_d in the same way that filenames are. An attacker could obtain the identification of the root directory, or even its blocks, if the last attack is successful. Directories are special files a small number of blocks. It is feasible a brute force attack to rearrange blocks on these small files, but the attacker still have to break down a symmetric encryption with K_f.

An attacker gets the metadata of a file. The first blocks that a user asks to SCFS are the metadata of a file. An attacker could put apart these blocks and cryptoanalise them to get K_{ff}. In this sense, SCFS ask for some random blocks apart from the actual ones to make this attack less feasible.

An attacker collects every block of a single file. If an attacker eavesdrops the communication of a user he is able to get the list of blocks that conforms a file without any need of decrypt the metadata. But since the IDA algorithm takes place after the encryption and the attacker has no information about the K_s that creates the vector space for the algorithm, he has to permute the blocks of the file at random and then perform the cryptanalysis to break down the file encryption with K_f. The IDA step adds an additional security to the encryption that makes feasible to use a weaker algorithm for encryption of the file for little devices, as RC4.

An attacker deletes a file of a user. Since SCFS does not offer a deletion mechanism as explained in section 4.7, this attack cannot be performed.

An attacker controls a group of nodes in the network. The original CFS uses Chord at the DHT level and the identifiers of each node depend on the network address. SCFS shares this behaviour with CFS, and then an attacker has a limited range of identifiers for his nodes. The DHT layer is is able to perform long jumps even if a segment of the ring. The buckets of Kademlia, that is the DHT used in SCFS, are even stronger against a ring breakage (Maymounkov and Mazières, 2002). Furthermore, since users only need k blocks of n to reconstruct the original file, the blocks stored in the segment that the attacker controls are not really needed. As stated in (del Campo et al., 2008b), SCFS does not currently have an economics system to detect and prevent malicious behaviour. A system of this kind minimises the dangers of malicious users in the network by means of identifying them as soon as possible.

Lazy nodes do not send a block, or an attacker sends a false block back. Nodes that are overwhelmed with links or prefer not to cooperate with other nodes won't answer to block requests. Since SCFS uses an IDA algorithm, users will only have to gather k out of n blocks to reconstruct the original file. Modification of a block is easily identified with the hash included in the iNode. An economics system to prevent such kind of attack is advisable, as stated previously.

A node try to decrypt the blocks than it stores. SCFS encrypts blocks using AES and breaks down files in pieces with the IDA. The information of a single block makes little sense to the node that stores a single block. Even the block identification contents no information about the contents, its filename or its publishers.

Any user overwrites one or more blocks of another user. Since there are many users in the network that store many files with many blocks, there are high chances that a legitimate user use the same identifier that another user for different blocks. Attackers may use the same identifier with malicious intentions. The DHT is able to store several blocks under the same identifier, and it will returns the whole list after a question. Since SCFS stores the hash of the blocks in the metadata, users can discriminate their blocks from other user's block even if they are stored under the same key. Since SCFS uses SHA, we assume that an attacker has not a feasible way to modify a block while keeping the same hash and he won't be able to

overwrite a user's block.

An attacker traces the publisher of a file. In this attack we assume that the attacker could overcome the attacks listed above. That assumption needs a considerable processing power, and only major government agencies can carry out such an attack. Since there is not an authentication mechanism in the system, the agency has not any knowledge of the original author of the file. In this sense the agency has to supplant the node that store the initial iNode and wait for the publisher to download the file to catch him. Even in this case, the network can use one of the mechanisms described in (Freedman and Morris, 2002) or (TOR,) to warrant access anonymity to its users.

A government agency prosecutes nodes that distribute a file. As stated in the previous attacks, nodes in SCFS do not know which kind of content they store, their contents or filename. In many countries, denying knowledge of the content that a node store is enough to prevent prosecution from local authorities to node administrators.

5 CONCLUSIONS

In this paper we analyse several distributed filesystems from the point of view of security and conclude that, as far as we know, there is not a satisfying solution to store personal data. We analyse the security requirements of such service and conclude that CFS is the network that best matches the necessities of personal users. Then, we present a Secure Cooperative Filesystem that solves the security problems of CFS. The ideas behind SCFS were implemented in (del Campo et al., 2008a).

As a result of a security analysis, we conclude that SCFS solves many of the security requirements of a distributed file system, but it has several possible enhancements. Particularly, further research must be done in order to include an economic system for SCFS.

ACKNOWLEDGEMENTS

This project was partially supported by a grant of the Spanish MCT, under the program Consolider-Ingenio 2010 CSD 2007-0004, under the ARES project.

REFERENCES

Tor anonymity online. Webpage: http://www.torproject.org.

(2000). Mnet-mojo nation. Web page: mnetproject.org.

Annapureddy, S., Freedman, M. J., and Mazières, D. (2005). Shark: Scaling file servers via cooperative caching. *NSDI*.

Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. (2000). Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009/2001 of *Lecture Notes in Computer Science*, page 46. Springer Berlin / Heidelberg.

Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2001). Wide-area cooperative storage with cfs. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA. ACM Press.

del Campo, J. V., Hernández-Serrano, J., and Pegueroles, J. (2008a). Scfs: lewis.upc.es/svn/dfs.

del Campo, J. V., Hernández-Serrano, J., and Pegueroles, J. (2008b). Securing cooperative file system. In *ESORICS (sent)*.

Dingledine, R., Freedman, M. J., and Molnar, D. (2001). The free haven project: Distributed anonymous storage service. *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, Berkeley, CA, USA, July 2000, Proceedings:*, 2009/2001:67.

Freedman, M. J. and Morris, R. (2002). Tarzan: A peer-to-peer anonymizing network layer. In *CCS*.

Grothoff, C., Grothoff, K., Horozov, T., and Lindgren, J. T. (2006). An encoding for censorship-resistant sharing. http://gnunet.org/.

Hasan, R., Anwar, Z., Yurcik, W., Brumbaugh, L., and Campbell, R. (2005). A survey of peer-to-peer storage techniques for distributed file systems. In *IEEE International Conference on Information Technology (ITCC)*. Las Vegas.

Maymounkov, P. and Mazières, D. (2002). Kademlia: a peer-to-peer information system based on the xor metrid. In *IPTPS*, pages 53–65.

Rabin, M. O. (1989). Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the ACM*, 36(2):335 – 348.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149 – 160. ACM Press.

Tatara, K., Hori, Y., and Sakurai, K. (2005). Query forwarding algorithm supporting initiator anonymity in gnunet. *Proceedings. 11th International Conference on Parallel and Distributed Systems*, Vol. 2:235 – 9.