



Project no: 507613

Project acronym: Euro-NGI

Project title: Design and Engineering of the Next Generation Internet, towards convergent multi-service networks.

Instrument: Network of Excellence

Thematic Priority: Broadband For All

**Deliverable reference number: D.WP.JRA.6.3.5**

**Deliverable Title: Protection of stored data by system level encryption schemes**

Due date of deliverable: 2005/05/31

Actual submission date: 2005/05/31

Start date of project: 1st December 2003      Duration: 3 years

Organization name of lead contractor for this deliverable: URM2 (31) and BTH (49)

Editor's name for this deliverable: Cristiano Paris and Markus Fiedler

Revision [final]

**Project co-funded by European Commission within the Sixth Framework Programme (2002-2006)**

**Dissemination Level**

<b>PU</b>	Public	X
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	



**Editor's name:** Cristiano Paris, Markus Fiedler

**Editor's e-mail address:** [paris@disp.uniroma2.it](mailto:paris@disp.uniroma2.it), [markus.fiedler@bth.se](mailto:markus.fiedler@bth.se)

With Contributions of the following partners:

Partner Num.	Partner Name	Contributor Name	Contributor email	Comments
31	URM2	Cristiano Paris	<a href="mailto:paris@disp.uniroma2.it">paris@disp.uniroma2.it</a>	-
40	UPB	Radu Lupu	<a href="mailto:radu.lupu@elcom.pub.ro">radu.lupu@elcom.pub.ro</a>	-
32	NTNU	Tonnes Breknw	<a href="mailto:tonnes@q2s.ntnu.no">tonnes@q2s.ntnu.no</a>	-
49	BTH	Markus Fiedler	<a href="mailto:markus.fiedler@bth.se">markus.fiedler@bth.se</a>	-
36	WUT	Wojciech Mazurczyk	<a href="mailto:W.Mazurczyk@tele.pw.edu.pl">W.Mazurczyk@tele.pw.edu.pl</a>	-

#### Version History

Version - 1.0 -	2005/05/31 -	- URM2 <a href="mailto:paris@disp.uniroma2.it">paris@disp.uniroma2.it</a>
First Public Release		

**Project Acronym:** Euro-NGI

**Project Full Title:** Design and Engineering of the Next Generation Internet, towards convergent multi-service networks.

**Type of contract:** Network of Excellence

**contract No:** 507613

**Project URL:** <http://www.eurongi.org>



# Contents

<b>1 Introduction</b>	<b>2</b>
1.1 Overview . . . . .	2
1.2 Mobility-related changes . . . . .	3
1.3 Off-line Security . . . . .	3
<b>2 Digital Rights Management</b>	<b>5</b>
2.1 Introduction . . . . .	5
2.2 Open issues and reactions . . . . .	7
<b>3 Current Solutions</b>	<b>8</b>
3.1 Encrypted Block Devices . . . . .	8
3.2 Pass-through models . . . . .	8
3.3 Digital Rights Management . . . . .	13
3.3.1 Encrypted Computation for Digital Right Management . . . . .	13
<b>4 New Approaches</b>	<b>15</b>
4.1 Integrated Pass-through models . . . . .	15
4.2 New DRM models . . . . .	16
<b>5 Conclusions</b>	<b>17</b>
5.1 Conclusions . . . . .	17
<b>References</b>	<b>18</b>

# 1 Introduction

## 1.1 Overview

Information leakage is recently becoming an important threat to privacy. The attention has been focused on the protection of data when stored in large information repositories. This led to the development of complex access control models which don't take into account data protection when information leaves the repository and is stored in a file system used by the client. This form of protection is known as *off-line security* and it's left completely to user's responsibility who is often not even aware of this duty.

Therefore a new model for access control taking into account client-side data distribution and protection must be developed. During the years a lot of work has been done in the field of device and file system encryption to ensure the privacy of data and protection against physical-access off-line attacks, in which an attacker has complete access to the physical device in which data are stored.

This solution proved to be effective but we still lack a mechanism to link the access control machinery in charge in the repository from which data are accessed to the encryption layer available at the client and possibly control it in some sense.

Other forms of data protection regard the possibility to allow the access to the information retrieved from a repository only through the use of a proper interface executed at the client terminal. When referring to files contained copyrighted stuff this kind of client-side data protection goes under the name of *Digital Rights Management (DRM)* which is somewhat of a misnomer for what is actually a generalization of access control. Whereas access control usually limits itself to the permission or prohibition of requests for operations from a limited set of operations, what is known as DRM concerns itself with the full range of legitimate usage concerns. This is because DRM concerns itself with enforcement of object-specific policies irrespective of where the object in question is currently located.

A central problem in DRM is therefore contract enforcement in a potentially malicious environment. Since the vast majority of objects that one seeks to distribute for legitimate usage are software or digital content objects, this problem is very similar, if not equivalent to the secure autonomous mobile agent problem.

Another central problem is that of usage semantics. To see why the problem of usage semantics is extremely difficult, if not impossible to solve in a DRM context, consider the case where a user makes a copy of a music CD in his/her possession. If the user keeps the copy in his/her archive as a backup copy, and does so for the entire future, that copy is legitimate, which means that the original copying operation was also legitimate. If, however, the user distributes the copy to a friend permanently, maybe even receiving payment for it, then that copy becomes illegitimate. The question is then: does the original action of copying become illegitimate because of this? In other words: to solve the semantics problem associated with legitimate usage, it seems necessary for computer systems to deduce users' intentions—also in the future. So what one needs to solve this problem unerringly, is a mind-reading, future-predicting integrated circuit.

## 1.2 Mobility-related changes

Recent years have experienced an explosive growth in the use of wireless and mobile devices. The mobile phone is not only used for calling, but also for sending and receiving messages such as SMS (Short Message Service) or MMS (Multimedia Messaging Service), simple surfing using WAP (Wireless Application Protocol) and increasingly for data services such as mobile e-mail. With the generation shift from GPRS (General Packet Radio Service) to UMTS (Universal Mobile Telecommunication System), the availability of higher data rates in mobile systems boosts new multimedia applications such as video calls and enables the use of laptop computers in a mobile environment as if they were connected to a fixed network. As a complement offering even higher data rates in limited areas, there is a growing number of WLAN hotspots infrastructure available. An increasing number of handheld computers and PDAs is nowadays equipped with GPRS and WLAN facilities. Thus, GPRS, UMTS and WLAN allow “people on the move” to communicate in a seamless “ABC” (Always Best Connected) manner. They have the possibility to access a multitude of services (e-commerce, e-banking, e-brokerage, e-government, e-payment, etc.) from their mobile terminals (almost) anywhere and anytime. These services and their enablers (positioning systems, payment systems etc.) constitute an important part of today’s telecommunication business as communication goes mobile.

Mobile (or wireless) communication devices have become an important part of many people’s daily life and a means of expressing themselves. New models offering interesting features and special designs are popular and show that their owner is up-to-date. As such, they are interesting and sometimes easy-to-grab loot. Mobile devices being used as “daily companion” use to carry plenty of user-related, highly personal and sensitive data such as contact information, personal messages, schedules etc. Mobile or handheld computers and the corresponding memory sticks and cards might also contain sensitive files and applications. In general, there is no special access control to such devices. The adversary just needs to plug the memory stick or insert the memory card to its device in order to be able to access and possibly mis-use the contents. Another risk applies to the accessibility to devices via unsecure WLANs; hackers might gain access to the file system this way. Obviously, security problems with unsecured/-protected file systems might compromise their user’s privacy, including non-classified data. Affected users might consider not using certain devices and services for certain tasks anymore, which might be counterproductive for the telecommunication business.

## 1.3 Off-line Security

*Off-line security* refers to the ability to protect data when it’s not being transferred through a network infrastructure. This usually means protecting data in the place where it’s stored, that in the storage device of a system or in its file system. Off-line security aims to prevent attacks in which the intruder has physical access to the device where data are stored. For a server this means protecting data stored in its hard drives or tape backups, thus for these systems off-line security have been identified with physical access control security.

Anyhow recently the problem has become even worse with the introduction of mobile devices which are likely to be stolen and, generally speaking, physically accessed more easily than servers. In addition the increasing introduction of global communications and always-on connection eased the access to data stored in remote servers for which controlling the access to data retrieved from them is an important issue to deal with.

Solutions have been proposed along the time and can be classified in two categories:

- *Encrypted Block-device.* In these solutions the storage device used by the file system is encrypted as a whole. This allows for simple and checkable implementations which can be used with almost any file system but comes at the cost of an almost-zero flexibility and performance hit.
- *Pass-through model.* In this approach the encryption layer is moved into the *Virtual File System (VFS)* or into a specific file system implementation. These solutions usually offer a greater flexibility and performance than those offered by encrypted block-device approach. Anyway we often lose file system independency.

Even if these approaches proved to be effective, they take into little account the key management issues. The burden of managing the security of the file system is often left to the end-user who is not competent or willing to take this responsibility.

## 2 Digital Rights Management

### 2.1 Introduction

So far, most part of the research activity focused on security services to protect against unauthorized access by a third party (a.k.a. Trudy) while data in transit through the network or stored; the data being completely under the control of the recipient (a.k.a. Bob). In addition, the new trend of electronic digitized content (audio, video, data) marketplace and e-commerce business models imposes the necessity to control further the legitimacy of actions the recipient can do over the data, according to a license agreement.

In this regard, Digital Rights Management (DRM) technology aims to provide copyright protection and ownership assertion.

To achieve these objectives, DRM relies on a large area of cryptographical techniques and security technologies as content authentication (e.g. watermarking, steganography), entity authentication (e.g. biometrics), encryption, authorization, digital signatures, security protocols and smart-card technology.

Further, in this chapter we present an overview of DRM goals, issues and main European Commission recommendations.

Nowadays, it is accepted by all stakeholders that a pervasive DRM service is possible only with widespread of inter-operational DRM technologies doubled by a legal framework to promote their deployment.

Stakeholders have a different view on interoperability, for instance, consumer/user want to be able to choose whatever devices he likes or DRM solutions; content generator/provider needs does not be constrained by the distribution channel, the device manufacturer that his products can be used with different content service.

The widespread of DRM technologies will provide the promised environment for the development of on-line legal content and services marketplace and allows for new business models to be deployed.

A workable copyright environment based on DRM would cope with digital rights definition, management and enforcement of usage rules in a flexible manner to allow exceptions with respect to particular cases (e.g. copies for private use).

DRM uses several technologies in order to provide for description and identification of digital content or for setting and enforcement of the usage rules established by right holders or by law for digital content.

The identification consists in the attribution of an identifier (e.g. ISBN, ISWC, ISRC or ISAN), similar to the legacy printed works, and marking with a sign (e.g. through watermarking).

Metadata is used to describe digital content (e.g. author, date of publication originating country) and to indicate which rights model to apply the content, the type of the fees or the period the access should be granted.

DRM systems use digital wrappers to protect the digital content, metadata and rules while transferred through the network.

DRM community state that to support interoperability and facilitate complex rules definition, the information associated to digital content must be described in a formal way using a digital rights expression language.

It was proposed several high-level languages, most of them inheriting expressivity and interoperability from

XML. Among the most important are:

- XrML (eXtensible rights Markup Language) provides a method to specify and manage rights and rules associated with a large kinds of resources (content/services). XrML is adopted and supported by a part of most important IT players like Adobe, Microsoft, HP Labs, XEROX or Moving Picture Group.
- ODRL (Open Digital Rights Language) is an open-standard language which provides a method of expressing rights and policies using flexible and interoperable mechanisms.

Other examples of emerging standards: *Extensible Access Control Markup Language (XACML)*, *MPEG Rights Expression Language*.

DRM is promoted by European Commission through the following activities:

1. issue standards and recommendations to support compatibility and interoperability between DRM solutions
2. build a legal framework and supervise the enforcement of the policies among the Member States
3. co-finance R&D projects (e.g. through IST Programme) to support a rapid development and deployment of a global and open infrastructure for IPR management based on open standards (request cooperation of stakeholders, discuss business models).

The *High Level Group (HLG)* on Digital Rights Management established by EC (as part of eEurope 2005 Action Plan) identified the following challenges [1]:

1. interoperability requirements, including standardization developments, for DRM to meet users' expectations
2. acceptance and trust by users with particular emphasis on security and privacy
3. the impact of DRM on existing rights management approaches, in particular the application of levies

The HLG's Final Report states the following principles and recommendations:

1. DRM must not become a commercial or technology licensing control point
2. DRM will be an open standard to guarantee interoperability and compatibility
3. DRM must fit business models not vice versa

The legal framework related to digital content IPR, in the EU, is defined by the Directive on the Harmonization of Copyright and Related Rights on the Information Society (2001/29/EC), which address the use of DRM (a.k.a. Copyright Directive). According to the Directive, Member States should provide legal protection for technological measures through its implementation, by the end of 2002.

To date, massive piracy activity hampered deployment of legitimate services, therefore besides better technical protection measures, it is needed anti-piracy laws enforcement.

The main factors on which depends the consumer migration to legitimate services are:

1. the number of devices implementing compatible DRM solutions which are confident to legitimate content

2. interoperability of media formats and DRM solutions from multiple vendors
3. the awareness of consumers/users about law infringement in the case of using or publishing illegally protected content/services
4. trust and confidence of users to use DRM-enabled hardware
5. the capability of DRM systems to provide an alternative for copyright levies; to ensure "fair compensation" for right holders in the case of private use of copies of the content; also, to enable forwarding of licensed and paid content between users (i.e. superdistribution)
6. the capability to supply consumer-friendly DRM while ensuring secure technical protection
7. the state-of-the-art in development of related services like secure payment systems, PKI development and deployment and trust infrastructure to provide overall security in a distributed environment

## 2.2 Open issues and reactions

So far, the work on DRM has been done individually by different stakeholders with different perspectives, which conducted finally DRM systems to fail interoperate and to meet the challenge of copyright protection while respecting the rights of the purchaser of the copy. In this way, it is expected that the copyright framework at EU (referred above) and international level will provide the legislative framework to solve this problem.

There exist some groups (such as Electronic Frontier Foundation) which consider that the current DRM solutions are infeasible. They highlight a number of (inherent) contradictions between DRM promoters and human behaviour, yielding customers vulnerable at the moment of equipment replacement or software upgrade and reduce flexibility of the digital content use.

The DRM content protection solutions which request remote control at the purchaser's computer system, raised a huge controversy among users and seem to infringe even the EU Directive 95/46/EC on the protection of individuals. The DRM opponents affirm that it could create a computer environment which cannot be trusted, regardless the legal merits of this controlling activity.

There are, also, some DRM issues to address which hamper the purchasers from exercising their legal rights, for instance as it is known after a defined period the copyright work becomes part of the public domain to anyone to use freely, but the DRM systems developed have not time dependent mechanisms to remove the control system after the term of copyright expires. Another problem is that even though copyright law does not restrict the resale of the copyrighted content, currently DRM technology has not support for this.

To sum up, the desired balance between the interests of right holders and purchasers, is still an open issue and for a successful design and deployment of DRM systems it is needed to cope with all these issues.

## 3 Current Solutions

### 3.1 Encrypted Block Devices

One of the oldest and proven technology for file system data encryption is represented by the so-called *encrypted block devices*. This approach consists in the encryption of the device the file system relies upon for storing its data. In most modern operating systems these devices are seen as single and large contiguous files, each composed out of distinct blocks whose length is generally set to 512 bytes (the minimal addressable block in the hard drive).

In order to encrypt the file system, a specific layer is put in the running kernel between the Virtual File System (VFS) layer and the block device management subsystem. The encryption layer accepts read and write requests from the file system module and encrypts/decrypts the data flowing to/from the block device management subsystem.

The main advantage of the solutions based on this approach is the simplicity of the implementation. The device to be protected is represented by a single file (the block device itself) so the security management model is minimalistic: it's a all-or-nothing encryption.

Most of the times the user has to provide a single passphrase which is used by the encryption layer to encrypt and decrypt data in the device. It's worth noting that the encryption key used to crypt a given block is specific being evaluated from the passphrase for that specific block and it differs from that used in a different block. This trick helps to prevent cypher-text based attacks since a block device can be several gigabytes long.

The simplicity of the approach is also its main drawbacks. The encryption layer is completely unaware of the file system machinery: the data encrypted/decrypted are viewed as raw and unstructured. This makes impossible to model complex security configurations in which, for instance, different level of security/encryption are needed or the files have to be encrypted with different keys belonging, for example, to different users of the system.

Performance usually suffers from this issue, since no file is allowed to be stored in clear in the file system. In addition the burden of the security management is left to the user which is completely out of control of the repository which ultimately provided the data stored in the file system.

Important examples of this approach are represented by the *encrypted loopback* devices and the *dm-crypt* module of the Linux Operating System.

### 3.2 Pass-through models

A completely different approach in the protection of file system data is represented by solutions based on the so-called *pass-through-models* in which the encryption of data is moved up into the hierarchy of the operating system, possibly into the VFS or in a specific file system. This approach makes the encryption layer aware of the data being encrypted and allows much more flexible security models to be implemented. The implementations of this model are often known as *Cryptographic File Systems*.

Functions of a cryptographic file system are the same as functions of a traditional file system, but apart from the traditional one everything is done in a secure manner.

We can classify cryptographic file systems in very different ways. Here we will focus on two classifications. First we can generally divide those systems into: cryptographic network file systems and cryptographic storage file systems. Cryptographic network file systems protect the information sent between a user's workstation and the file server. For instance a good example is SFS (Secure File System) which is covered later in this chapter. This cryptographic file system stores plaintext on the file server, but protects the link to the client. Cryptographic network file systems are appropriate when the file server is trusted not to disclose or alter stored data.

On the other hand, cryptographic storage file systems keep files encrypted on the file server. The users need not trust the file server to protect confidentiality. The idea of applying encryption to data stored in the file system is not a new one. There are many examples in the literature and the commercial world of file systems that utilize encryption. We will focus mainly on cryptographic storage file systems later in this chapter.

Secondly we can divide cryptographic file systems based on their approach to encrypting a file system. We can point out five different types:

- Block based systems encrypt one disc block at a time. They operate below the file system level. No knowledge of file system (being encrypted) is required. Such systems can write to raw device or preallocated file. Also, they do not reveal information about individual files (such as sizes and owners) or directory structure. Examples: SFS (Secure File System), BestCrypt, CryptoLoop, CGD (Cryptographic disk driver), vncrypt, vnd.
- Disk based systems encrypt data at the file system level. These file systems have access to all per-file and per-directory data, so they are able to perform more complex authorization and authentication than block-based systems. Additionally disk-based file systems can control the physical data layout. This means that disk-based file systems can limit the amount of information revealed to an attacker about file size and owner. Examples: EFS (Encryption File System, [2]), SFS (Steganographic File System, [3]), StegFS [4],[5].
- Network loopback based systems can not control the on-disk layout of files. They have two major advantages: they can operate on top of any file system and they are more portable than disk-based file systems. However major disadvantages are performance and security. Since each request must travel through the network stack, more data copies are required and performance suffers. Security also suffers because NBFS are vulnerable to all of the weaknesses of the underlying network protocol. Examples: CFS, TCFS.
- Stackable file systems can operate on top of any file system. These file systems are a compromise between kernel-level disk-based file systems and loopback network file systems. Stackable file systems can operate on top of any file system. However they do not have to copy data across the user-kernel boundary or through the network stack. Examples: Cryptfs [6], Ncryptfs [7].
- Application based encryption: applications like gpg or crypt allow users to encrypt/decrypt their individual files. File is in cleartext on the disk while the user is editing and saving it. This solution, however, is quite inconvenient for users. Each time they want to access a file, users must manually

decrypt or encrypt it. The more user interaction is required to encrypt or decrypt files, the more often mistakes are made, resulting in damage to the file or leaking confidential data

Along the years many implementations of cryptographic file systems have been presented. Here's a non-exhaustive list of the most widely known cryptographic file systems:

**BestCrypt.** BestCrypt is a commercially available loopback device driver supporting many ciphers. Such a loopback device driver creates a raw block device with a single file, called a container, as the backing store. This device can then be formatted with any file system or used as swap space. Each container has a single cipher key. The administrator creates, formats, and mounts the container as if it were a regular block device. BestCrypt is ideal for single user environments but unsuitable for multiuser systems. In a single-user workstation, the user controls the details of creating and using a container. In a multi-user environment, however, the user must give the encryption key to a potentially untrustworthy administrator. Moreover, the ability to share containers among groups of users is limited, as BestCrypt gives different users equal rights to the same container.

**Secure File System.** The Secure File System (SFS)[8] implements a cryptographic storage file system for MS-DOS. Despite its underlying operation system, SFS has a surprising number of creative and useful features. The primary goal of SFS is to protect bulk data stored on a disk. Peter Gutmann developed SFS until 1995. Since SFS operates entirely on a single machine, the user must only trust the local operating system and hardware. No concept of a superuser exists in the early versions of MS-DOS. SFS protects against disclosure of data to unauthorized persons who may gain physical access after files are encrypted. The most interesting feature of SFS is the emergency access mechanism. To safeguard against data loss, this mechanism can recover lost passwords. The emergency access mechanism employs Shamir's secret sharing scheme in which trusted escrow agents receive  $n$  key fragments. Any  $m$ -sized subset of the  $n$  agents can recover the key. However, no smaller subset can feasibly recover the key.

SFS uses the Cipher Feedback (CFB) mode of the Message Digest Cipher with the Secure Hash Standard (MDC/SHS) encryption algorithm. The password generates an intermediate key by iterating a one-way hash function over the password several hundred times. This intermediate key can decrypt the master key which in turn decrypts/encrypts the data on the disk.

Currently all development on SFS has ceased.

**EFS (Encryption File System).** EFS was found in Microsoft Windows, based on the NT kernel (Windows 2000 and XP). It is an extension to NTFS and utilizes Windows authentication methods as well as Windows ACLs.

EFS is designed to be transparent to the end-user under normal circumstances. When a user attempts to access one of their encrypted files, the EFS will locate the private key used to encrypt the File Encryption Key that encrypted the file, decrypt the File Encryption Key, and decrypt the file. If an attempt is made to access a file encrypted by another user, EFS will fail to find the private key that can be used to decrypt the File Encryption Key and the user will be presented with an "Access Denied" condition.

Though EFS is located in the kernel, it is tightly coupled with user-space DLLs to perform encryption and the user-space Local Security Authentication Server for authentication. This prevents EFS from

being used for protecting files or folders in the root or winnt directory. Encryption keys are stored on the disk in a lockbox that is encrypted using the user's login password. This means that when users change their password, the lockbox must be re-encrypted. If an administrator changes the user's password, then all encrypted files become unreadable.

**StegFS.** StegFS is an extension of SFS (Steganographic File System) which was designed by Designed by Adi Shamir. It is reportedly more elaborate and stable than SFS. These systems attempt to hide the information in such a way as to discredit its very existence and they combine steganography with encryption. If adversaries inspect the system, then they only know that there is some hidden data. They do not know the contents or extent of what is hidden. This is achieved via a modified Ext2 kernel driver that keeps a separate block-allocation table per security level. It is not possible to determine how many security levels exist without the key to each security level. When the disk is mounted with an unmodified Ext2 driver, random blocks may be overwritten, so data is replicated randomly throughout the disk to avoid loss of data.

**Cryptographic File System (CFS).** The Cryptographic File System (CFS) [9] introduced a relatively robust cryptographic storage system for the UNIX operating system and it is most known and important cryptographic file system. It was developed in 1993 by Matt Blaze from AT&T Bell Laboratories. CFS interesting feature is that it pushes confidentiality into data storage. Although several other file systems had already incorporated cryptography to secure network file transmissions, CFS is the first well-documented UNIX cryptographic storage file system. CFS investigates the question of where encryption should be placed in a file system: at the low-level hardware or the user-level. Several goals guided the CFS design including the issues of key management, transparency, and portability.

In a manual file encryption system, the user inputs a password or key whenever encryption or decryption takes place. Instead of requiring a password for each encryption operation, CFS asks the user once per login for a password. This password acts as a seed to compute a key stream. CFS eliminates the problem of reentering passwords, but can generate new difficulties in password management. A user may need to remember several passwords to protect unrelated directories. Transparent performance and access semantics allow users to perform necessary tasks without knowledge or interference from the underlying cryptography. That is, one should not notice significant differences from a traditional UNIX file system. CFS succeeds in hiding most of the cryptography. When using one encryption password, CFS does not incur noticeable performance penalties after an initial pause to compute the key stream. Moreover, access semantics work as in a normal UNIX file system. However, simultaneous use of several passwords will cause many key streams to be loaded into memory. In the extreme case, CFS can cause thrashing by filling up all memory with key streams.

CFS provides end-to-end encryption from the client back to the client. All encryption operations take place on the client machine. The server is trusted to reliably store and retrieve information and not to alter storage. The user must completely trust the client machine and anyone who can gain root access on the client machine.

CFS protects data, file names, and symbolic links from disclosure outside the trusted computing base. It offers no protection against unauthorized modification. The goals of the CFS design do not include emergency key recovery or protection against denial of service. CFS employs the DES block cipher in Output-Feedback and Electronic Codebook (OFB+ECB) mode to create a key stream from a password.

Because passwords directly encrypt files, CFS has the advantage that only the holder of the password can access files. No key ring waits to be stolen by an adversary. However, this makes re-encryption and emergency access difficult. No native password changing procedures exist. Emergency access or key escrow could help recover lost passwords, but it would also give adversaries an easier task to acquire a key.

**TCFS – Transparent Cryptographic File System.** The Transparent Cryptographic File System (TCFS)[10] is improved security model that was designed in CFS. TCFS was developed at the Università di Salerno in Italy in 1997. TCFS works fundamentally the same as CFS, except that TCFS is an in-kernel implementation. One of the principle tenets of TCFS is not to trust the server except to store files. When a client machine attempts to access an encrypted file, the encrypted file blocks are sent over the network. Upon arrival, the blocks are decrypted on the client machine. TCFS extends the UNIX file attributes to include an encryption bit. When this bit is set, TCFS protects the file with encryption.

As is with CFS, the metadata are not encrypted. Although some confidentiality may be lost to file sizes and file names, existing utilities will continue to operate normally. The documentation does not explain whether mechanisms exist to protect against unauthorized modification. Each user's standard login password decrypts a key ring to access files. Therefore, all files with the encryption bit set are encrypted by the owner's key. No built-in data loss protection exists. TCFS currently supports compile-time options for ciphers including DES, IDEA, and RC5. Keys are stored in a global key file which can be accessed by a user's password.

TCFS provides transparent management of user keys. Instead of the user being prompted to enter a key by a specialized program, random keys are generated by the file-system. The random keys are encrypted with the password of the user and stored in a file called `/etc/tcfspasswd`. The user id is used as an index in to the key file to retrieve the key for the user process trying to access the file. This has the advantage of transparently handing the encryption of the file data. At the same time, it reduces the security provided by the system to the difficulty of decrypting the user password to retrieve the keys. TCFS provides a finer granularity of encryption than CFS. An extended file attribute called `secure` is tested upon the creation of the file. If the `secure` attribute is set, subsequent read and write operations are directed through the cryptography layer. If the `secure` attribute is not set, the read and write operations are treated as normal.

A recently released version of TCFS implements a group secret sharing protocol. A file can be opened only when a certain threshold number of group members log in simultaneously. All group members must log into the same workstation. This is not similar to the concept of group sharing in the UNIX file system. TCFS has a strong following, but is somewhat less robust and well-documented compared to CFS. It has several weaknesses that make it less useful for deployment. First, the reliance on login passwords as user keys is not safe. Also, storing encryption keys on disk in a key database further reduces security.

**Cryptfs.** Cryptfs is the stackable, cryptographic file system and part of the FiST toolkit. It is implemented as a kernel resident file system. Cryptfs can be mounted on any local or remote directory and can utilize any underlying file system such as UFS, ext2 or NFS. Cryptfs does not require any specialized daemon program to run as it layers itself on top of the existing underlying file systems. Cryptfs is implemented as a stackable code interface. Similar to CFS, users of Cryptfs are prompted to enter a pass phrase to generate a key for authentication. A message digest of the pass phrase is generated using MD5 and is

stored in the memory used by Cryptfs. The keys to encrypt the file data are not stored in a file, this makes it more secure than TCFS. As the user must enter the pass phrase at the start of each new session that is started, this results in reduced flexibility but provides greater security.

Cryptfs uses Blowfish symmetric-key cryptography for encryption of file data and meta-data. A 128-bit key provides a balance between performance and encryption strength. Keys can be used in two different ways. The performance of Cryptfs is improved by the location of the encryption/decryption layer in the kernel. This results in the same number of context switches as file access in a regular file system.

**NCryptfs.** NCryptfs is an improved version of Cryptfs. Its primary goal is to ensure data confidentiality while balancing security, performance and convenience. NCryptfs is a security wrapper that binds to a directory that stores ciphertext data. The ciphertext directory may be on any file system (e.g., EXT2 or NFS). Through NCryptfs, a cleartext view is presented via the standard UNIX file access API. We provide convenience by allowing administrators and users to customize the behavior of NCryptfs, while picking sensible defaults.

The threat models NCryptfs addresses include network sniffers, untrusted servers, and stolen machines. Normally, when exporting file systems over the network, cleartext data is sent over the network and the server must be trusted to keep the data confidential. When NCryptfs is deployed on the clients, only ciphertext file data is sent over the network, and the server does not have access to the cleartext data. Corporations and governments are storing more and more sensitive data on laptops, which are often stolen along with their valuable data. When NCryptfs is used, a stolen laptop will not reveal useful information to the thief (see Section 1.2). In both of these scenarios, NCryptfs attempts to restrict the information a compromised system reveals to an attacker to just the information that is actively being used.

NCryptfs has support for multiple users, multiple keys, multiple ciphers, and multiple authentication methods (including challenge-response authentication between user processes and the kernel). It is able to operate on Ad-hoc groups, allowing users to delegate "join" privileges to others, and for others to join or leave groups as needed. It works on per-process and per-session keys, with hooks for processes to be informed of certain activities in NCryptfs. This cryptographic system file has key timeouts and revocation, which in addition to per-process keys allows to suspend and resume processes based on key validity, as well as to add encryption transparently to unmodified programs that have already begun running.

## 3.3 Digital Rights Management

### 3.3.1 Encrypted Computation for Digital Right Management

As pointed out in Section 2.1, DRM for software and digital rights management is very similar, if not identical to the secure autonomous mobile agent problem. A mobile autonomous agent tasked with carrying out some action on behalf of its sender, possibly after gathering information is a typical example of proposed agent applications. Such an agent is effectively a migrating process, that communicates with the hosts it visits. Each such host typically needs a support environment for such agents, which could be called the *host platform*. The agent is sent by some party, called the *sender*. An agent of this type can be of two types:

- the agent that is a true migrating process which carries along with it data, or
- an agent embedded in data, which is supposed to facilitate the use of the data by the host platform in some manner.

An autonomous mobile agent must be able to compute decision results, and act on them without communicating with its sender. If the agent needs to send a message to its host platform, where the contents have been derived using secret data (as when a digital signature is generated), those secret data must also be protected during storage and use. If the computation of a decision result depends on the use of secret data carried along with the agent's code, then those data must be protected in storage and during use. Furthermore, the result of the decision must still be useable by the agent.

There exist several systems for so-called secure multi-party computations. Many solve very specific multi-party computation problems. There is, however, a feature common to almost all of these systems: any given protocol completion can only compute at most a finite number of functions that require input of fixed size. They are therefore at best limited to computing in parallel several primitive recursive functions.

Thus a secure autonomous mobile agent must secure its workspace, as well as its program code. It must also be capable of authenticating itself in order to access resources it is authorized to use. To accomplish this, it is necessary to develop a cryptographic system that makes it possible to generate agent code that:

1. is executable in encrypted form;
2. can process its workspace in encrypted form; and
3. can communicate with its environment in either plaintext or ciphertext.

A system for encrypted computation presented by Brekne in [11] solves these problems in principle, but requires cryptographically strength for long computations, and an impractical space complexity. A central question is therefore whether it is possible to construct a general method, which has lower space-complexity.

## 4 New Approaches

### 4.1 Integrated Pass-through models

The main issue with the revised solutions is that the security of the file system in which data are stored is under the complete responsibility of the user. This means that the source from which data are possibly retrieved has no means to control information leakages from the client terminal.

Usually data are strongly protected when inside repository: this is an easy task since data are accessed through few well-defined interfaces which can be deeply analysed and secured. With this respect centralization of data storage in the repository is the key feature for a successful protection.

On the other hand, when data are transferred from the source repository to the client terminal, the access controls that were used in the centralized repository get out of scope. This leaves data completely unprotected from the risk to be exposed or accessed by people not allowed to see or write them then. This situation is even worse in a mobile scenario in which the client terminal is often represented by a tiny device which is likely to be stolen.

Therefore there should be a way for the controls used in the repository to be extended to the local clients. The only way this can be accomplished is through the use of the concept of *delegation*. In this architecture the repository's controls delegate the protection of data when stored inside the client's file system to the client's system itself.

This delegation is possible specifying the security requirements which should be fulfilled by the client's system to protect data properly. Conversely the client provides the list of its security capabilities that can be exploited to protect data. At this point the client and the repository negotiate the set of controls that must be applied before the data transfer actually happens. After the negotiation phase the data is transferred to the client.

An example of this approach can be the requirement stated by the repository to have the local file system wipe the received data from the client terminal as soon as they are no more accessible from the network repository, i.e. the data can be stored as long as the user is connected to it. This kind of approach leverages completely the user from the burden of managing security on his/her side.

Along with the specification of the security requirements the repository should state their scope of application. With this respect the only model suitable to support this approach is the pass-through model in which the encryption layer is aware of the file system structure and can selectively apply the proper controls depending on the position of the retrieved files in the file system.

The implementation of this approach requires the specification of a description language which the security requirements needed for safely storing the data in the client-side file system can be specified through. This description language should include statements about:

- The desired algorithms and key-length which data should be encrypted by.
- The maximum period of time data should reside in the client file system.

- User authentication techniques (password based, token based and so on).
- Access control policy definition and enforcement requirements.
- The scope which the controls fulfilling these requirements must be enforced.
- The persistence of the controls.

The list of controls negotiated with the remote repository to fulfil the security requirements should be included in the meta data of the files in the file system. Depending on the persistence of these controls, the list can be stored either in the client system's main memory (temporary controls) or in the storage device which the file system relies upon. In order to grant the maximum interoperability among repositories, client systems and the different file system implementation, it's highly recommended to develop an open standard.

## 4.2 New DRM models

Encrypted computation is a potential new model for DRM, which has yet to materialize with the efficiency desired. Recent results by Barak [12] indicate that non-black-box computation is inherently more powerful than black-box computation. This fits well with the observation that Brekne's system [11] is not a black-box system. The usual definition of a black-box system requires that the cryptosystem at least be a group homomorphism, if not a ring homomorphism. The system by Brekne is in general not homomorphic, and is thus all negative results relating to such cryptosystems do not apply here. Thus there seems to be hope that a system offering encrypted computation with both acceptable time-complexity and space-complexity can be constructed, although it is not obvious how to do so.

It is, however, somewhat misleading to focus only on encrypted computation. There are other promising methods, which seem capable of at least contributing to the solution of the DRM problem. Schneider describes in [13] a fundamental access control model called execution monitoring.

Denote by  $\Psi$  the set of all possible terminating and non-terminating executions for any single process. Elements in  $\Psi$  are strings where each symbol represents an event, a state, or a combination of these. Denote by  $\Psi_p$  the associated set of possible executions for a process  $p$ . Define a security policy as a set  $\mathbf{S} \subseteq \Psi$ . A process  $p$  adheres to  $\mathbf{S}$  if all of  $p$ 's executions,  $\Psi_p$ , are contained in  $\mathbf{S}$ . In practice,  $\mathbf{S}$  should be definable by a first order logic predicate. Sets definable by such predicates are also called *properties*. Sets that cannot be defined by first order logic are called *quasi-properties* for the duration of this article.

The monitor observes events, states, or a combination of these during  $p$ 's execution. These actions form a string  $\sigma$  which is a prefix of an execution  $\psi \in \Psi_p$ . If there is no execution in  $\Psi_p \cap \mathbf{S}$  having  $\sigma$  as prefix,  $p$  is terminated.  $p$  is considered to have violated the security policy by attempting an execution not in  $\mathbf{S}$ . This mechanism is one way of blocking or limiting undesirable event sequences during a process' execution.

## 5 Conclusions

### 5.1 Conclusions

In this document we have examined the different aspects of the protection of data stored in a file system. As we have seen this is becoming an increasing issue in recent times when we experience the use of mobile devices and pervasive communications. We presented and analysed two forms of data protection:

- *Off-line security* which aims to protect data from unauthorized physical access. Most of the times this means protecting data from anyone but the legitimate owner of the client terminal.
- *Digital Rights Management* which aims to protect the data stored in the client terminal against access using unauthorized interfaces executed in client terminal itself, that is protecting data from the owner of the client terminal.

We presented the different approaches and solutions found in literature and in particular we pointed out that almost none of the presented implementations take into account security management aspects which are left completely to user's responsibility. We have considered an integrated approach in which the source from which data are retrieved checks and delegates the data security management by passing the user who is leveraged from this burden.

## Bibliography

- [1] High Level Group on Digital Rights Management, “Final report,” Tech. Rep., March-July 2004. 2.1
- [2] Microsoft, “Encrypting file system for windows 2000,”  
<http://www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp>, 1999. 3.2
- [3] Ross Anderson, Roger Needham, and Adi Shamir, “The steganographic file system,” in *Information Hiding, Second International Workshop*, 1998, pp. 73–82. 3.2
- [4] H. Pang, K. Tan, and X. Zhou, “Stegfs: A steganographic file system,” in *Proceedings of the 19th International Conference on Data Engineering*, 2003, pp. 657–668. 3.2
- [5] A. D. McDonald and M. G. Kuhn, “Stegfs: A steganographic file system for linux,” in *Information Hiding*, 1999, pp. 462–477. 3.2
- [6] E. Zadok, I. Badulescu, and A. Shender, “Cryptfs: A stackable vnode level encryption file system,”  
<http://www.cs.columbia.edu/library>. 3.2
- [7] C. P. Wright, M. Martino, and E. Zadok, “Ncryptfs: A secure and convenient cryptographic file system,” in *Proceedings of the Annual USENIX Technical Conference*, 2003, pp. 197–210. 3.2
- [8] P. C. Gutmann, “Secure filesystem (sfs) for dos/windows,”  
<http://www.cs.auckland.ac.nz/pgut001/sfs/index.html>, 1994. 3.2
- [9] Matt Blaze, “A cryptographic file system for unix,” in *Proceedings of 1st ACM Conference on Communications and Computing Security*, 1993, pp. 158–165. 3.2
- [10] G. Cattaneo, G. Persiano, A. Del Sorbo, A. Cozzolino, E. Mauriello, and R. Pisapia, “Design and implementation of a transparent cryptographic file system for unix,” <http://tcfs.dia.unisa.it>, 1997. 3.2
- [11] Tonnes Brekne, *Encrypted Computation*, Ph.D. thesis, Department of Telematics, Norwegian University of Science and Technology, 2001, Number 2001:67. 3.3.1, 4.2
- [12] Boaz Barak, *Non-Black-Box Techniques in Cryptography*, Ph.D. thesis, Department of Computer Science, The Weizmann Institute of Science, January 2004. 4.2
- [13] Fred B. Schneider, “Enforceable security policies,” technical report, Dept. of Computer Science, Cornell University, Ithaca, New York 14853, 1998, Revision of July 24, 1999. 4.2