

Secure data storage



Problems (1/3)

- ▷ The classical file system protection is limited
 - ♦ Physical protection assumptions
 - Physical confinement of storage devices
 - ♦ Logical protection assumptions
 - Access control performed by systems managing the devices
 - e.g. operating systems
 - Proper use of ACLs or other authorization mechanisms



Problems (2/3)

- ▷ There are numerous scenarios where this protection is useless
 - ♦ Direct/physical access to storage devices
 - Mobile computational units
 - Laptops, PDAs, smartphones
 - Removable storage devices
 - Tapes, diskettes, CDs, DVDs, memory cards
 - ♦ Bypassing of logical access control mechanisms
 - Unethical access by powerful users (e.g. administrators)
 - Personification of users



Problems (3/3)

- ▷ Distributed access raises security issues
 - ♦ Trust in (unknown) administration teams
 - ♦ Secure communications
 - Confidentiality, integrity
 - ♦ Remote authentication of users
 - Security level provided
 - i.e. how hard it is to impersonate someone
 - Integration among clients and servers
 - Applications, operating system
 - Interaction model
 - Sessions vs. requests
 - Entities
 - People vs. machines/systems



Solution:

File encryption

- ▷ Encryption/decryption of files' contents
 - ♦ Can safely circulate along dangerous networks
 - ♦ Can safely be stored in insecure storage devices
 - Either mobile or administrated by others
- ▷ Problems
 - ♦ Data retrieval
 - End-users cannot loose encryption/decryption keys
 - Illegitimate end-user encryption
 - Corporate data
 - ♦ File sharing
 - It implies some sort of (group) key sharing
 - ♦ Interference with regular storage administration procedures
 - e.g. backups



Ideal architecture (1/2)

- ▷ Cipher/decipher transparency
 - ♦ At the application level
 - ♦ At the level of OS file caches
 - But tacking into consideration authorization issues
- ▷ Visibility of securely stored data
 - ♦ Visual awareness
 - Of what is protected and not protected
 - ♦ Automatic setting of encryption attributes
 - With customization options

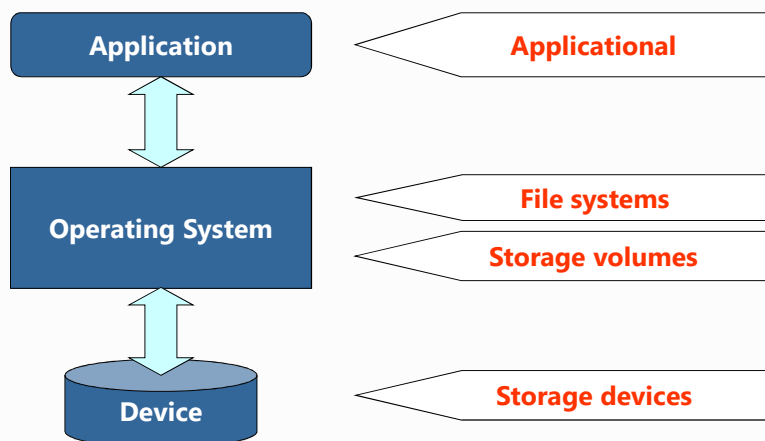


Ideal architecture (2/2)

- ▷ Easy sharing of encrypted data
 - ♦ By groups of users
- ▷ Decryption capacity under special circumstances by authorized people
 - ♦ Legal enforcement
 - ♦ Protection against the loss of decipher keys



Current approaches



Applicational

- ▷ Data transformed by autonomous applications
 - Little or no integration with other applications
 - Usually it is clear what is secure or not
 - e.g. using specific file extensions
- ▷ Data can be transformed with different algorithms
 - Adds flexibility, increases security
 - Complicates recovery procedures
- ▷ There are vulnerability windows
 - Cleartext resulting files used by other applications
- ▷ Hard to share data without sharing keys
 - Secret keys or public keys
- ▷ Examples:
 - PGP, AxCrypt, etc.



Storage volumes / devices

- ▷ Cipher/decipher operations at the volume / device level
 - Total transparency for applications and possibly to the OS
 - The visibility of protected data has volume / device granularity
 - Not required to handle file systems issues
 - Protection of meta-information and file data
 - Users and access rights
- ▷ Cannot differentiate accesses by different users
 - More suitable for personal storage devices
- ▷ Cannot solve issues raised by distributed file systems
 - Decipher occurs when data is fetched from devices to server caches
- ▷ Examples:
 - PGPdisk, LUKS (Linux Unified Key Setup)
 - Self-Encrypting Drives



Secure file systems: Approaches

- ▷ Data is transformed in the path between storage devices and the memory of applications
 - ♦ Storage device ↔ file cache
 - No protection for remote accesses (server deciphers)
 - The access to caches gets more complex
 - Coordination with ACLs
 - Knowledge of cipher/decipher keys by the OS
 - ♦ File cache ↔ memory of applications
 - Protection for remote accesses (clients decipher)
 - Can take place outside the OS (e.g. STDIO in UNIX)
- ▷ Examples:
 - ♦ CFS (Cryptographic File System), encfs
 - ♦ EFS (Encrypted File System)



Secure file systems: Limitations (1/2)

- ▷ File system integrity must be preserved
 - ♦ Some file attributes cannot be hidden
 - For keeping the regular file system operation
 - Because of other administration tools (e.g. backup tools)
- ▷ Attributes that can easily be hidden
 - ♦ Arbitrary file/directory names
 - Encrypted versions must conform FS naming rules
 - ♦ File contents
 - Preferably without changing file's size



Secure file systems: Limitations (2/2)

- ▷ Attributes that **cannot** (should not) be hidden/changed
 - ♦ **Object types**
 - They define the structure of the file system
 - ♦ **Contents of directories**
 - ♦ **Some well-defined names**
 - e.g. "." and ".." in UNIX
 - ♦ **Dates**
 - For managing backups
 - ♦ **Dimension**
 - For knowing the real occupation of storage devices
 - ♦ **Ownership**
 - For managing storage quotas
 - ♦ **Access protection**
 - For keeping the normal access control policies



Secure file systems: Practical encryption issues

- ▷ Uniform random access to encrypted data
 - ♦ **Ciphers with feedback are not suitable**
- ▷ Confidentiality
 - ♦ **Not advised to use the same key for different files**
 - Similar patterns could reveal similar files
 - ♦ **Not advised to use the same key for an entire file**
 - Similar patterns along a file could reveal its semantics
 - ♦ **Stream ciphers are not advised w/ the same key for different files**
 - Known-plaintext attacks could reveal contents of other files



CFS (Cryptographic File System)

▷ NFS extension

- OS ⇔ local CFS server ⇔ local or remote NFS server
- The NFS interface is kept
- The MOUNT interface changes
 - Includes a password

▷ Encryption / decryption operations

- Performed by the local CFS server
 - Files circulate encrypted in the network
 - Decrypted file contents are maintained in the client OS file cache
 - All local users with READ access to the file can read the decrypted contents
- Cipher/decipher keys supplied per each mount point
 - Communicated to the local CFS server by a modified mount command
 - This command uses the new MOUNT interface



CFS

▷ Encrypts file names and file contents

- Using two keys (K1 and K2) derived from a password

▷ Name

- Concatenated with and integrity control value
- Encrypted with ECB

▷ File contents

- Stream with OFB ⊕ block ECB
 - OFB with K1
 - ECB com K2 (disk blocks are not increased)
- OFB mask computed with K1 per mount point
- Random IV per file
 - Applied between XOR with OFB mask and ECB
 - Stored in the i-node GID
 - CFS provides the directory GID instead of the file GID



EFS (Encrypted File System)

- ▷ Windows NTFS extension
 - First appeared in Windows 2000
 - Provides encryption facilities to NTFS 5
- ▷ Functionality
 - Each user is bound to an asymmetric key pair
 - Stored and managed by the OS
 - Each file is encrypted with a unique symmetric key
 - FEK (File Encryption Key)
 - An encrypted file can be accessed by many users
 - For each file EFS stores copy of FEK encrypted with the public key of each authorized user
 - Encrypted FEKs are stored in a STREAM associated to the file
 - NTFS files are formed by sets of STREAMS
 - Each encrypted file is clearly visible
 - Using the Explorer file navigator



EFS cryptographic technology

- ▷ Algorithms
 - Asymmetric encryption of FEKs: RSA
 - Symmetric encryption with FEKs: DESX
 - $\text{DESX} \equiv \text{DES with whitening}$
 - $\text{FEK} = (K1, K2, K3)$
 - $C = K1 \oplus \text{DES}(K2, P \oplus K3)$
- ▷ Problems
 - Asymmetric key pairs are stored in disk
 - Loss risk
 - Illegitimate access by administrators
 - Files are decrypted by servers
 - No network protection for files stored remotely

