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

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

Secure communications

- Confidentiality, integrity

Solution: File encryption

Encryption/decryption of files' contents

- Can safely circulate along dangerous networks
- Can safely be stored in insecure store devices
  - Either mobile or administrated by others

Problems

- Data retrieval
  - End-users cannot loose encryption/decryption keys
  - Illegitimate end-user encryption
    - Corporation data
- File sharing
  - It implies some sort of key sharing
- Possible interference with regular storage administration procedures

Ideal architecture

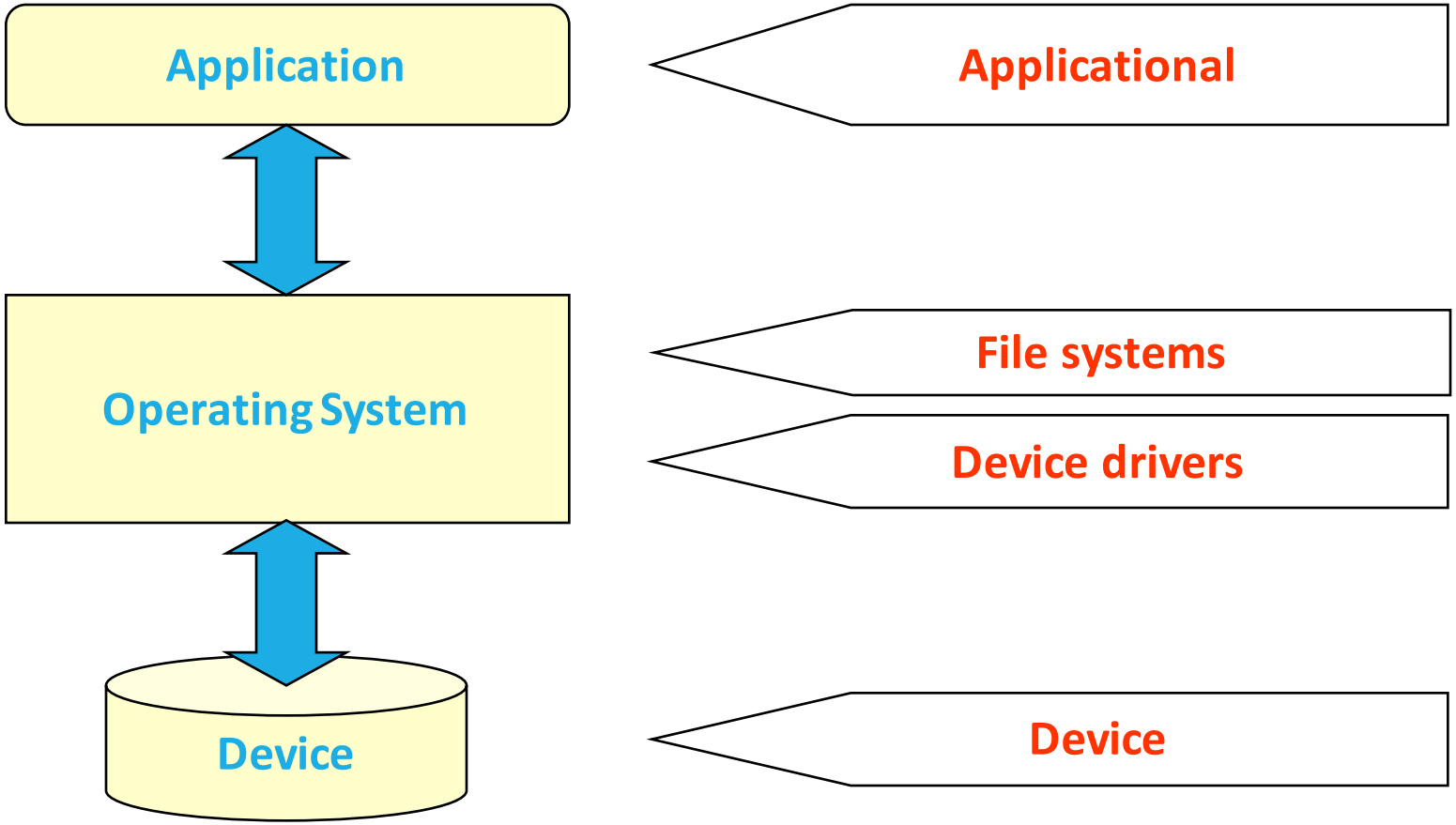
1. **Cipher/decipher transparency**
   - At the application level
   - At the level of OS file caches
     - But tacking into consideration authorization issues

2. **Visibility of securely stored data**
   - Visual awareness of what is protected and not protected
   - Availability of mechanisms for automatic setting of encryption attributes

3. **Easy sharing of encrypted data**
   - By groups of users

4. **Decryption capacity under special circumstances by authorized people**
   - Legal enforcement
   - Protection against the loss of decipher keys

Current aproaches



Application Level

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

There are vulnerability windows

- Cleartext resulting files for being used by other applications

Data can be transformed with different algorithms

- Increases security
- Complicates recovery procedures

Hard to share data without sharing keys

Examples:

- PGP, AxCrypt, TrueCrypt, etc.
- Also… RAR, ZIP

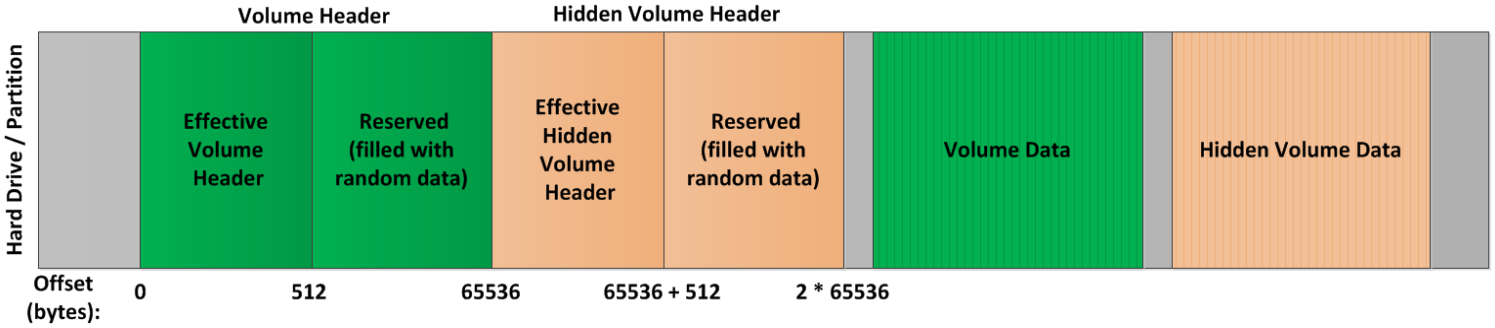
Application Level TrueCrypt

Creates file in the filesystem to emulate a storage volume

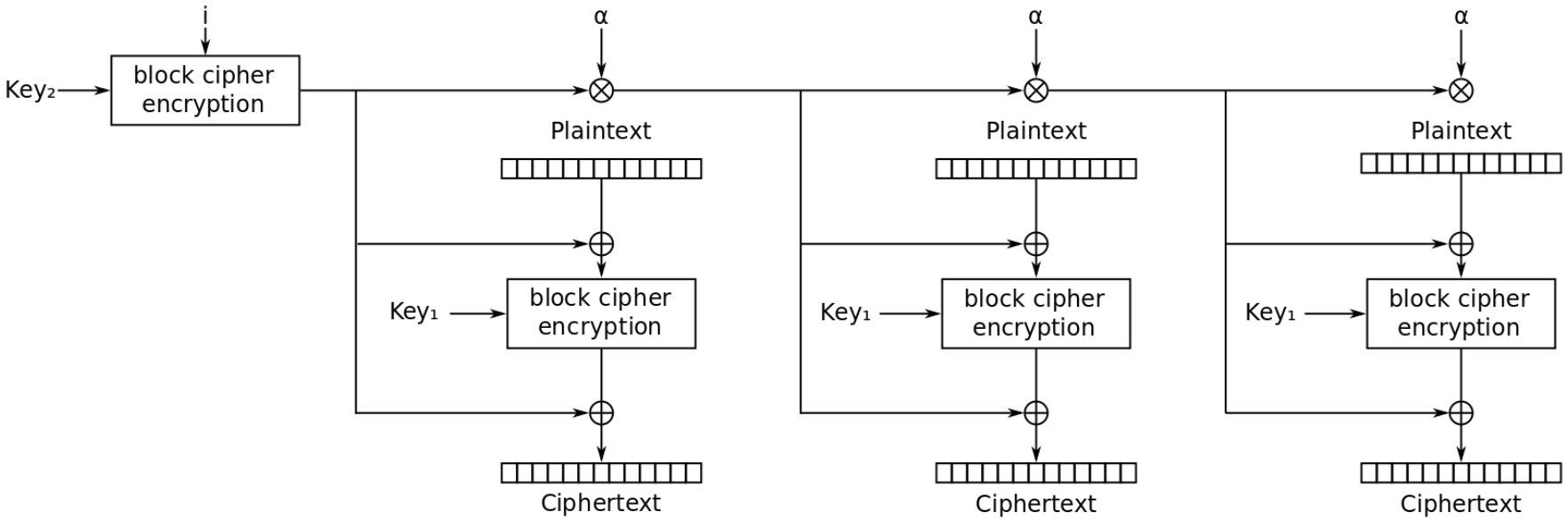
- Strong ciphers in cascades (e.g. AES+Twofish)
- AES-CBC, then AES-LRW, then AES-XTS
- Keys created with PBKDFS2 using SHA-512 and 2000 rounds

Supports Plausible Deniability

- Files have no clear text headers
- Files can have one or more volumes
  - Impossible to determine how many volumes there are



Application Level TrueCrypt



XEX with tweak and ciphertext stealing (XTS) mode encryption

Device drivers/OS Level

Cipher/decipher operations at the device driver level

- Total transparency for applications and to the OS
- The visibility of protected data with device granularity

Cipher policies made at the driver or application level

- 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

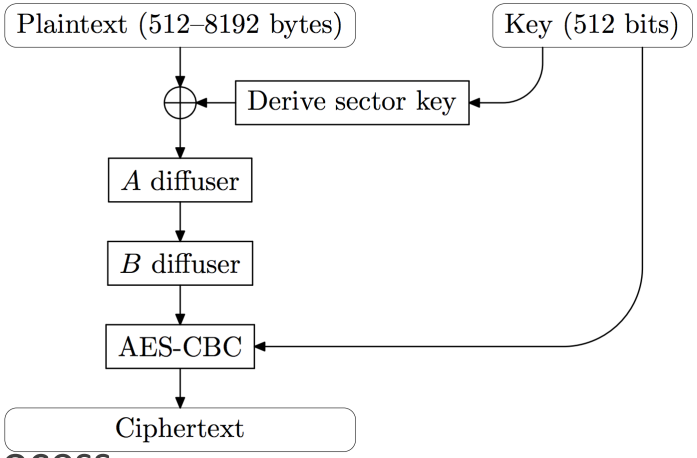
Cannot solve issues raised by distributed file systems

- Decipher done by the OS when data is fetched from devices to server caches

Examples:

- PGPdisk, LUKS, BitKeeper, FileVault

OS Level: BitLocker (Windows)



Encrypts an entire volume

- Small volume for starting the BitLocker boot process
- Full Volume Encryption Key -> K_{AES} and $K_{Diffuser}$

Key storage

- Volume stores FVEK, encrypted by Volume Master Key (VMK), encrypted by Key Protector Key
- Key Protector Key encrypted by user password, or protected in hardware module

Encryption process

- AES-CBC with 128 or 256 bit keys, applied to each sector, no MAC, no feedback
- IV = E(K_{AES}, e(s)) where e maps sector number to 16bit value
- Sector Key = E(Ksec, e(s)) | k E(Ksec, e'(s))
  - e' = same as e but last byte is 128
- Elephant Diffuser: Diffuses bits according to $K_{diffuser}$ (Deprecated)

OS Level: BitLocker

Without Diffuser CBC error propagation model enables Malleability attacks

- Change one bit (ASM JE into JNE)
- Can be exploited for Arbitrary Code Execution
- Diffuser has been deprecated

AES Key remain in memory as long as volume is mounted

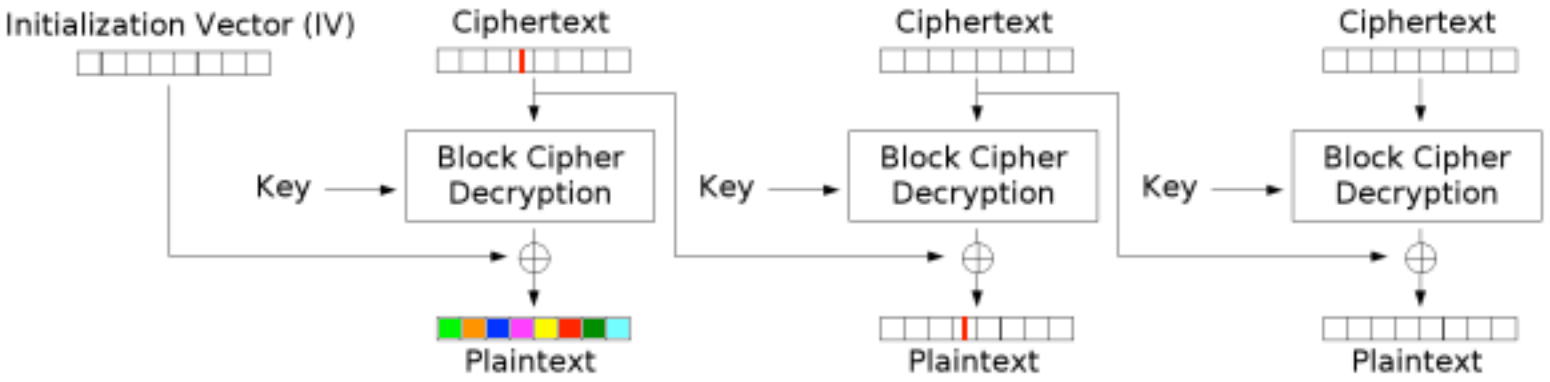
- Extraction is possible from RAM or Hibernation file

Possible to present a fake prompt and steal the password

- Authenticated bootloader can avoid this

OS Level: BitLocker

Malleability attack in CBC



Cipher Block Chaining (CBC) mode decryption

Device Level

Device applies security policy internally

- Upon boot, device must be unlocked
  - By providing the correct credentials
- Hardware implements cipher algorithms

Advantages

- No performance penalty
- Security applied by a secure model
- Difficult to extract data or even keys (like a smartcard)
- Possible to coordinate process with applications

Disadvantages

- Once device is unlocked, all data is accessible
- Security is limited by the algorithms supported by the Device

Device Level Self Encrypting Disks

Disk presents two different volumes

- Shadow Disk: Read Only, ~100MB; Contains unlock software; Freely available
- Real Disk: Read Write, contains user data; Protected

Two keys

- KEK: Key Encryption Key (Authentication Key)
  - Supplied by user
  - Hash stored in Shadow Volume
- MEK (or DEK): Media (Data) Encryption Key
  - Encrypted by the KEK

Boot Process

1. Bios sees Shadow Disk and boots from it
2. Code Validates user supplied password with Hash(KEK)
3. If validated, disk deciphers MEK into its RAM and Disk geometry is updated

Device Level
Secure Digital cards

Physical Write Protect lock

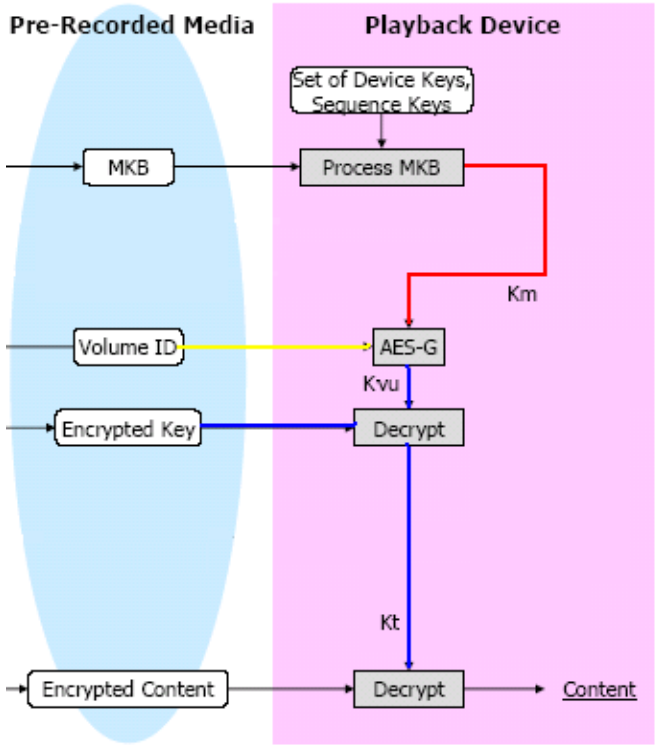
- Side of card

Content Protection for (Pre) Recordable Media (CPPM/CPRM)

- DRM mechanism supported by all cards
- Media Identifier: Identifies the content
- Media Key Block: Key to access content

Three Areas

- Security Area: Holds protection information: MKB, Media Identifier, Secuirty Status, etc...
- Protected Area: holds protected content
- Unprotected Area: holds freely available data



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 SO
- File cache ⇔ memory of applications
  - Protection for remote accesses (clients decipher)
  - Can take place outside the SO (e.g. STDIO in UNIX)

Examples:

- CFS (Cryptographic File System)
- 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 be hidden

- Arbitrary file/directory names
  - Encrypted versions must conform FS name rules
- File contents
  - Preferably without changing file's size

Secure file systems: Limitations (2/2)

Attributes that can (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
  - Full decryption for reading the last byte
  - Full encryption after updating the first byte

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 when using 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

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

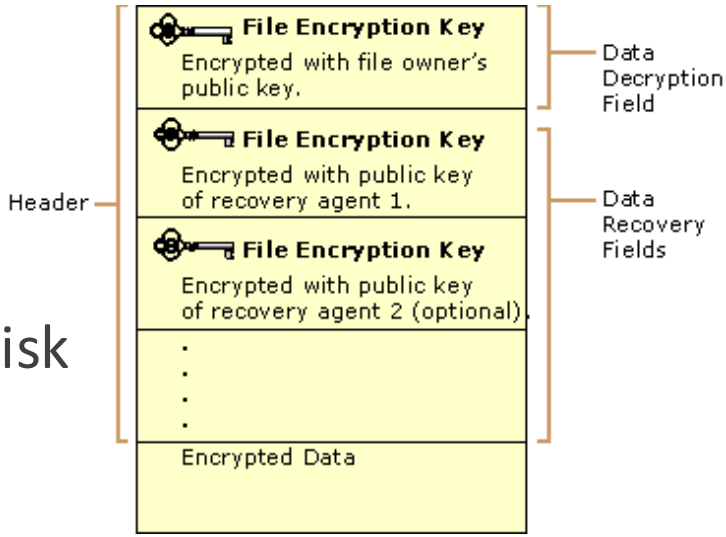
EFS cryptographic technology

Algorithms

- Asymmetric encryption of FEKs: RSA (1 FEK Per file)
- Symmetric encryption with FEKs: DESX
  - DESX ≡ DES with whitening

    FEK = (K1, K2, K3)

    C = K1 ⊕ DES(K2, P ⊕ 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