

A collage of various postage stamps and postmarks in red, yellow, and green ink, overlaid on a light yellow background. The stamps feature text in multiple languages, including 'PAR AVION', 'MADRID', 'COSTA RICA', and 'POSTAGE'. There are also circular postmarks and some numerical markings.

Encrypted File Systems

Don Porter
CSE 506

Goals



- ✦ Protect confidentiality of data at rest (i.e., on disk)
 - ✦ Even if the media is lost or stolen
 - ✦ Protecting confidentiality of in-memory data **much** harder
- ✦ Continue using file system features without losing confidentiality
 - ✦ Example: Backup
- ✦ Low overheads (space and CPU)
- ✦ Change keys and perhaps different keys for different data

Two major approaches



VFS

- ✦ Block device encryption
- ✦ Transparently encrypt entire partition/disk below the file system

ext4

Encrypted block device

- ✦ Linux: dm-crypt
- ✦ Windows: BitLocker
- ✦ Mac: FileVault 2

Generic block device

Block encryption intuition



- ✦ File system is created on a virtual block device
- ✦ Low-level read of virtual block device:
 - ✦ FS requests a block be read into page cache page X
 - ✦ Map to block(s) on real device
 - ✦ Request that blocks be read into a temporary page Y
 - ✦ Decrypt page X into page X
 - ✦ Return to file system
- ✦ Similarly, writes encrypt pages before sending to disk

Two major approaches



VFS

Encrypted FS

ext4

Generic block
device

- ✦ File System encryption
- ✦ Encrypt data between VFS/Buffer cache and low-level file system
- ✦ Linux: eCryptFS
- ✦ Windows: EFS
- ✦ Mac: FileVault 1

File-based intuition



- ✦ Idea: Mount a layered file system over a real one
- ✦ Application writes encrypted file 'foo'
- ✦ Encrypted FS opens real file foo
 - ✦ Stores some crypto metadata (like the cipher used) at the front
 - ✦ Encrypts pages in page cache, transparently writes at an offset

File-based intuition



- ✦ Read of file 'bar'
 - ✦ Encrypted FS asks real FS for file 'bar'
 - ✦ Uses metadata + secret key to decrypt
 - ✦ Stores decrypted pages in page cache
- ✦ Challenges:
 - ✦ Managing private keys
 - ✦ Enforcing read protection on decrypted data in page cache

Pros/Cons of disk encryption

✦ Pros:

- ✦ Hides directory structure, used space, etc
 - ✦ Metadata matters!
- ✦ Can put any file system on top of it

✦ Cons:

- ✦ Everything encrypted with one key
 - ✦ Encryption provides no confidentiality between users on a shared system
- ✦ Data must be re-encrypted before send on network
- ✦ Encryption overhead for public data (like /etc/hostname)

Vs. FS encryption



✦ Pros:

- ✦ Per-user (or per directory or file) encryption
- ✦ Only encrypt truly secret data
- ✦ Possibly send an encrypted file across network; use key (sent separately!) to decrypt on remote host

✦ Cons:

- ✦ Harder to hide/obfuscate directory structure and metadata
- ✦ More keys to manage
- ✦ Possibly easier to steal keys (debatable---harder to use TPMs)

Challenges



- ✦ Key management
- ✦ Read protection of live data
 - ✦ Swapping
- ✦ Booting the OS

Key management



- ✦ Or, where do we keep the secret key?
- ✦ Not in the file system!
 - ✦ There is a bootstrapping problem here
- ✦ Ideas?

Trusted Platform Module



- ✦ New hardware extension – common on PCs in last few years
 - ✦ Either on motherboard or in CPU chip itself
- ✦ Provides two useful features:
- ✦ **Measured Execution:** Basically, checks that the booted code (BIOS, bootloader, OS) match a given hash
 - ✦ Useful to detect tampering with your software
- ✦ **Sealed Storage:** Store a very small amount of data in non-volatile memory in the TPM chip
 - ✦ Only accessible from code with hash that wrote it

TPM Idea



- ✦ Store the private key for the file system in the TPM's sealed storage
- ✦ Only the trusted BIOS/bootloader/OS can access the decryption key
 - ✦ The drive alone gets you nothing!
 - ✦ Tampering with the OS image (on disk) to dump the disk contents gets you nothing!

Small problem



- ✦ Motherboard or CPU dies, taking TPM with it
- ✦ How to decrypt your files then?
 - ✦ BitLocker: As part of initialization, allow user to print a page with the decryption key. Put this in a safe place (not laptop bag)

Key management in FS-level encryption

- ✦ Each user has a key chain of decryption keys
 - ✦ Kernel is trusted with these keys
- ✦ On-disk, keychain is encrypted with a master key
- ✦ Master key is protected with a passphrase
 - ✦ That just happens to be the logon credentials
- ✦ So, with a user's passphrase, we can decrypt the master key for her home directory, then decrypt the keyring, then the home directory

Challenge 2



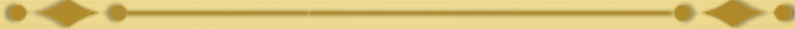
- ✦ The unencrypted data in the page cache needs to be protected
- ✦ If I encrypt my home directory, but make it world readable, any user on the system can still read my home directory!
- ✦ Encryption is no substitute for access control!

Swapping



- ✦ Care must be taken to prevent swapping of unencrypted data
 - ✦ **Or keys!**
 - ✦ If part of the file system/key management is in a user daemon, unencrypted keys can be swapped
- ✦ One strategy: Swap to an encrypted disk
- ✦ Another strategy: Give the encrypted file system hooks to re-encrypt data before it is written out to disk
 - ✦ Or put the swap file on the encrypted FS
- ✦ Subtle issue

Challenge 3: Booting



- ✦ You can't boot an encrypted kernel
- ✦ Decryption facilities usually need a booted kernel to work
- ✦ Big win for FS encryption: Don't encrypt files needed for boot
- ✦ Disk encryption: Usually puts files needed for boot on a separate (unencrypted) partition

Summary



- ✦ Two main types of encrypted storage:
 - ✦ Block and file system encryption
- ✦ Understand pros and cons of each
- ✦ Understand key challenges:
 - ✦ Key management
 - ✦ Swapping
 - ✦ Booting