CS 1699 Privacy in the Electronic Society

William Garrison
bill@cs.pitt.edu
6311 Sennott Square
http://cs.pitt.edu/~bill/1699

11: Disk encryption

Who decides whether stored data can be accessed?

Trusted code determines access control

- Ways around local gatekeeper?

Full-disk encryption offloads (some) enforcement to cryptography

- Coldboot attacks
- “Evil maid” attack
- TPM protections and further attacks

Mobile encryption

Traditionally, access control is handled by the operating system

Before processing system call to open a file, check that the requesting user is allowed

- This generalizes to other resources (e.g., network socket) as well

How does it decide?

- List of approved users per file? List of allowed permissions per user?
  - ACL vs. capabilities—we'll discuss these and other ways later
- Can all applications access everything the user can?
  - Traditional OSs: usually; Mobile: no

We've discussed many vulnerabilities this term, why trust this?

- If I don't trust the OS, who can I trust?

Relevant design principles

Saltzer and Schroeder 1975 proposed 8 design principles for secure systems

- We've directly/indirectly discussed many already

Most relevant principles:

- Economy of mechanism: Simple and small design
- Fail-safe defaults: Deny by default
- Complete mediation: Check every access
- Open design: Attacker knows the system
- Least privilege: Force processes to operate with minimum access needed

So, principled design = trust?

What can go wrong?

Bugs

- Improvements: Keep the trusted code small, formally verify if possible

Compromise OS

- Malware, kernel module/driver
  - Improvements: General “hygiene”
- Malicious insider?

Subvert OS

- Rootkit, boot from external media
  - Improvements: Secure the bootchain (coming soon)
- Remove hard drive?

Full-disk encryption attempts to give users more control

Idea: Encrypt sensitive data in addition to the trusted gatekeeper

- Without the key, data cannot be read; destroy key to delete data

How might encrypting the disk protect against the previously-mentioned attacks?

- Bugs, malware, kernel module, boot from external, steal drive

How might an encrypted volume still be vulnerable?

Details of VeraCrypt (successor to TrueCrypt)

Format of a volume, all encrypted except salts:

- 64-byte salt
- “VERA”
- Minimum version needed
- CRC-32 checksum of master keys
- 16 bytes of 0s
- Size of volume
- CRC-32 checksum of all above metadata
- Master keys
- Data
- Backup header (with different salt)

How are keys derived?

Salt is 512 bits (2^{512} keys per password)

- Offline attack?

User selects hash function for HMAC used in PBKDF2

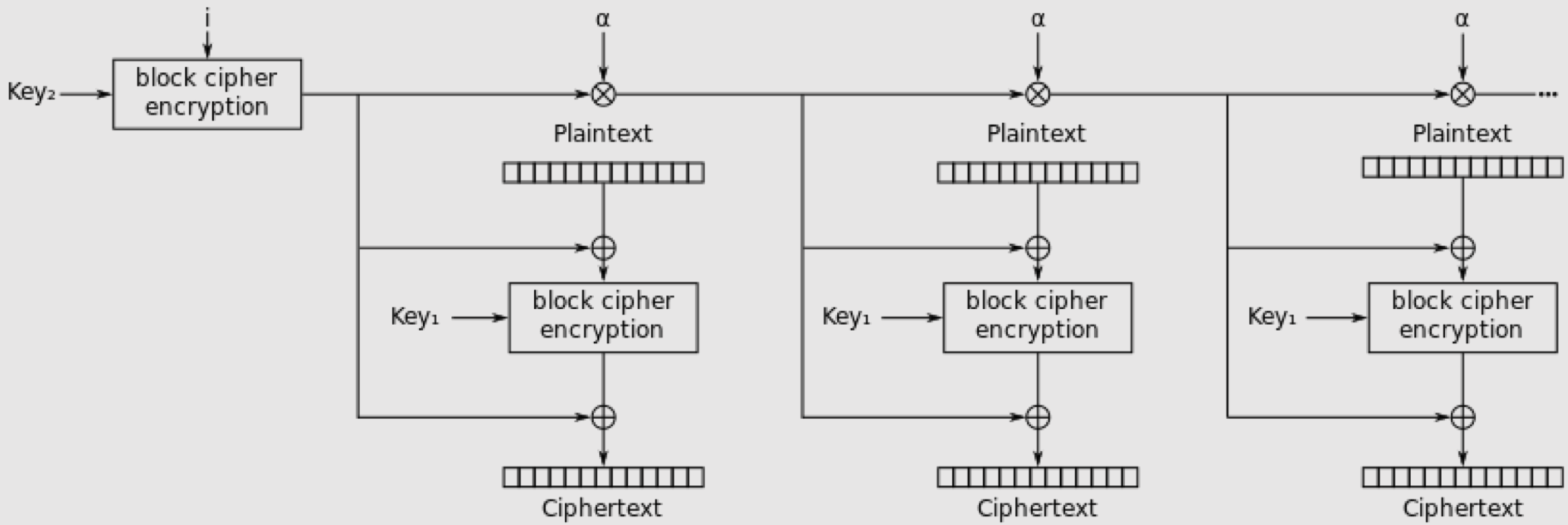
- SHA-512, SHA-256, RIPEMD-160, Whirlpool
- Default iterations for SHA-256: 200,000 for boot partitions, 500,000 for others
  - Users can adjust this when creating volume
- For multiple keys, derive large key and split (one key cannot reveal others nor password)

When initially booting, authentication is needed to read from the drive

1. Read the first 512 bytes (header)
2. Prompt for password, derive keys
   - Use unencrypted salt from header
3. Attempt to decrypt header
   - Which hash?
   - Which encryption algorithm? AES, Twofish, etc.
4. If you see “VERA”, check the CRC-32
5. If passes, decrypt volume keys and reinitialize
   - XTS block mode

What code does this? Where is it? Is it encrypted?

XEX mode (very similar to XTS)



XEX mode encryption

What is the security model, here?

What if we use “hibernate” mode?

In short, this writes the contents of RAM to disk in case of power loss

- Can this be encrypted?
- What’s stored here?

An attacker can retrieve keys from this region!

Coldboot attacks: RAM isn't as volatile as we pretend it is!

In 2008, Halderman et al. showed that DRAMs actually hold their contents for seconds to minutes

- If cooled, even longer
  - -50 C achievable with duster spray, 10 min.
  - -196 C with liquid nitrogen, hours

Then what?

What about Trusted Platform Module?

- How can it help?
- Why doesn't it solve the problem?

“Evil Maid” attack seeks to retrieve keys even if the machine is shut down

Recall that a minimal OS must prompt for passwords and initialize decryption

Consider the following:

1. Attacker gains physical access to shut-down computer
2. Attacker writes a hacked bootloader and leaves
3. User boots, enters password, decrypts
4. Key is sent to attacker!

Can we prevent this?

- Trusted boot!

Trusted boot, intuition

Idea: Only boot "trusted" code

… Trusted by whom? Who decides?

High-level: Ship machine with first level of trust, which verifies the next level, etc.

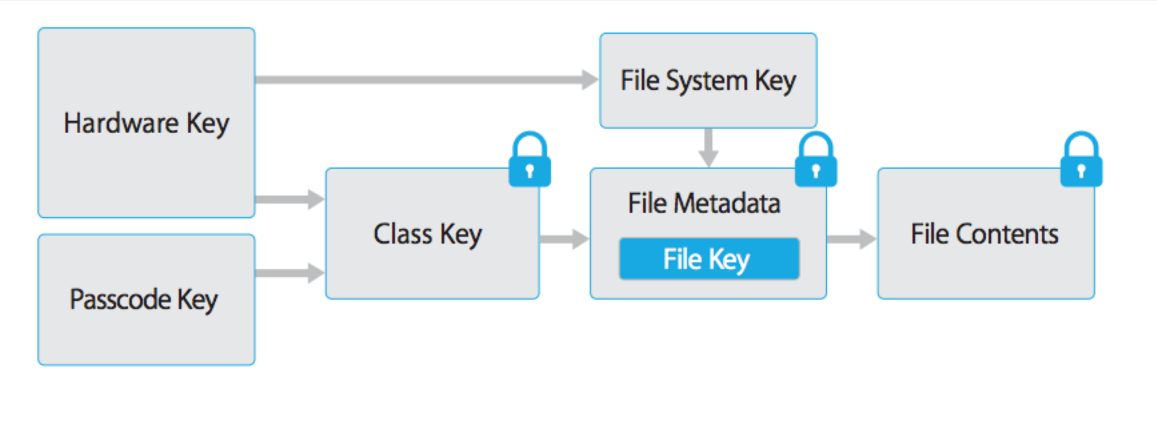
- e.g., BIOS (or equivalent) checks bootloader, which checks kernel, which checks system files
- Verify signature and/or checksums
  - Difference? Which in what scenarios?

Full Disk Encryption (FDE) vs. File-based Encryption (FBE)

FDE: Encrypt at the disk sector level (below FS)

FBE: Encrypt at the file level (above/within FS)

iOS uses FBE with per-file key stored in file metadata



Using FBE, iOS can provide different levels of access

Complete protection: Can only be accessed while unlocked

- How?

Protected until first user authentication: Can only be accessed after user unlocks after first boot

No protection: Can always be accessed

Developers choose which files fall into which category

Conclusions

OS generally enforces access control

- This is okay, but needs additional components to be secure for most threat models

Physical access is still a hard threat to overcome

Disk encryption is useful, but not without limitations

- Subtle to do correctly
- Many threats not avoided

Next time: How does the OS store and check permissions?