# Lecture 19 - iOS

## Security

6.1600 - fall 2023

MIT

Corrigan-Gibbs & Zeldovich

**Today:** Capstone lecture for platform security module.

We will look at the design of iPhone/iOS platform

**Goal:** Understand how concepts we have seen in this module (isolation, software sowain, secure boot) show up in a deployed system.

---

## Life cycle of iPhone

* Turn on phone
* Install & run apps
* Buy stuff
* Leave phone at restaurant
* Get phone stolen

... Will cover sec mechanisms at every step.

When you are learning about security defenses, good to remember:

* Some defenses are primarily there to protect HW vendor's business interests.

  e.g. DRM?

* Some defenses are primarily there to protect the user's interests (and indirectly Apple's)

  e.g. Encryption at rest?

* Sometimes, these interests are aligned.
* Sometimes, case is less clear. (e.g. App store? Trusted boot?)

# Turn on Phone: iOS Secure Boot

- Goal: Make sure that Apple-signed OS kernel is running.
  - ↳ Protects against persistent malware, people from installing custom/open OS on phone, someone from tampering w/ OS


- BootRom checks sig on LLB, runs
- LLB checks sig on kernel, runs
- → Failure = recovery mode.

- Very similar to what Nickolai described w/ PS3

**Idea:** Have special "secure enclave" that holds key — decrypts it only if user enters correct PIN.

↳ Surprisingly challenging to do safely

- Secure enclave is its own processor, also uses secure Boot
  - + Uses "measured boot" to derive secret encryption key that depends on OS being run — can't tamper w/ enclave OS & get data
- Secure enclave generates long-term secret UID on first boot & stores w/ fuses
  - ↳ uses UID to encrypt files
- Enclave communicates w/ secure storage over enc channel (they have shared secret)
- Secure storage holds:
  - enc key for user data
  - hashed passwd (hashed w/ UID)
  - Counter

## People "jailbreak" their iPhones... How?

By defeating secure-boot process.

- Bugs in BootRom and LLB could allow booting non-Apple OS

e.g. Checkra1n

↳ BootRom (burned into HW) code provides support for loading code via USB ("DFU mode")

e.g. if you really mess up OS & LLB code and can't boot

↳ Problem: Writing USB drivers is not so easy, iPhones w/ A9 chip have a UAF in BootRom

↳ Via USB, can trick phone into executing arbitrary OS w/o verifying signature.

CAN'T UPDATE BOOTROM — Can't fix!

But, doesn't persist across restarts.

→ Clever patch on more recent iPhones... maybe have time to discuss later.

↳ Patch later subverted.

With checkra1n attack, user can exploit bug in app processor CPU to install any OS on app processor.

Recall: Apple can't Patch BootRom on device

Clever idea: On newer phones (A10+), secure enclave detects when phone has booted in DFU mode (over USB) and panics on attempt to access user data

→ Even if you can't patch app proc, can patch enclave!

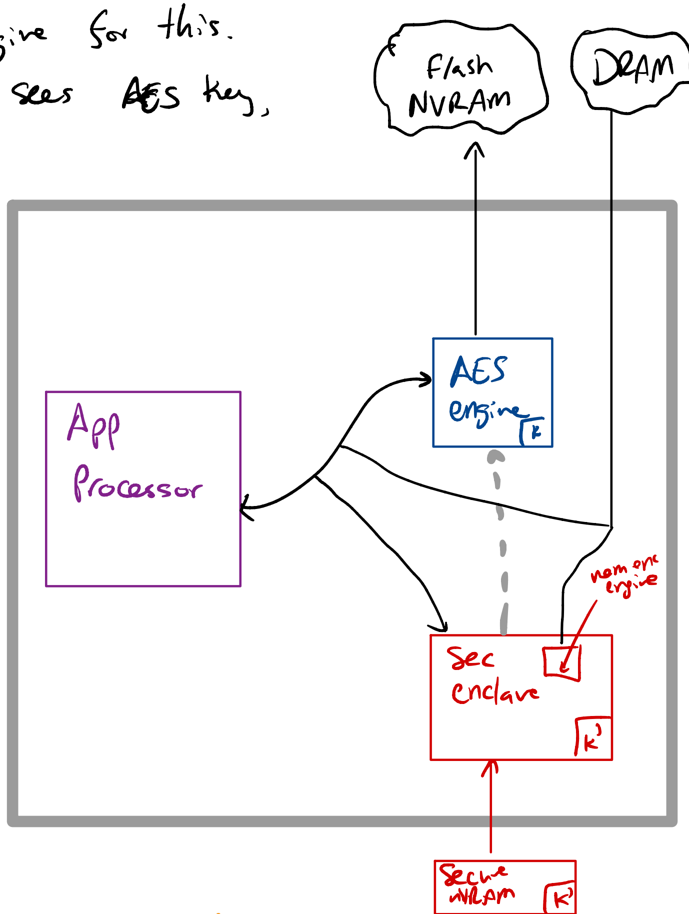
→ On A10, there's a bug in enclave too that allows reading enclave memory.

# Unlock phone: iOS Protection for Data at Rest

Threat: Someone takes your phone & wants to get the data off of it.

Basic idea: Encrypt all data w/ 128-bit AES key.

* Special hw AES engine for this.
* App processor never sees AES key, so can't easily leak to OS or apps...
  - ↳ Even if you compromise OS, can't get raw keys.
  - ↳ OS can't accidentally leak key.




## But, where do we store AES key?

(Can't use 4-/6-digit PIN as key... too short!)

(Can't store key in normal flash)

IDEA: Store PIN in enclave
↳ Still hashed w/ slow (80 ms) hash fn.

## PIN Auth w/ sec enclave

1. User enters PIN.
2. iOS passes PIN to enclave
3. Enclave enforces timeout
4. Enclave passes hashed PIN to sec storage
5. Sec storage checks PIN
   - ↳ If correct: Return AES key, zero guess ctr
   - ↳ Else: Increment guess ctr
     - ↳ If too many guesses, erase key
       - ← special hw support for erasure "effaceable storage"
6. Enclave passes AES key to AES engine (bypassing app processor)
7. App processor sends data to AES engine to be decrypted

Defeats many attacks:

- Brute force ✗
- Replace secure enclave w/ backdoored one ✗
- Guess PIN & reboot ✗

→ Similar strategy to ensure erasure of other important pieces of data (CC#, FaceID, ...)
Remote wipe of device

What about TouchID / Face ID?

↳ Always need PIN on reboot.
↳ otherwise, keys stored in enclave.
- To make it harder to swap out TouchID sensor while device is running (& feed in recorded fingerprint), enclave & ID sensor share a secret.

→ For max security, you'd power down device

→ Secure enclave + secure storage make PIN-based encryption much harder to break.

* So far, we have discussed how to go from PIN ⟶ AES key.
* Once device has root AES key, builds a tree of other keys...

Plan
* Main file-system key (kept in effaceable storage)
* "Class key" for type of protection
* Each file encrypted with own key

- Different levels of protection ("class")
  - No R/W when locked (keys on lock)
  - Append when locked (sk wiped on lock, but pk left around)
    - ↳ Useful for writing msg to user when phone locked
  - R/W when booted ⟶ Default for app data
  - No protection (but still encrypted to allow remote wipe)

# Where could bugs remain?

- Kernel on app processor is big. → Bugs!
- Even though it doesn't see AES keys, it sees lots of sensitive info (CC#, PIN, passwd)
  - ↳ many exploits
- Boot code on both processors may have bugs
- Could extract secrets using h/w attacks — probes, power analysis, etc. $ $ $
- Could steal secrets via "side-channel attacks"
  - ↳ Having separate AES engine likely makes stealing AES keys difficult
  - ↳ Still could steal secrets on device.

⋮

Install app: **App Security**

- Need to download & run software written by random people on the Internet. → Malware risk!

Security plan:

PC: You're on your own (essentially any app can see your files)
↳ Multiple users but still...

Browser: Don't let JS access sensitive data (isolation)

iOS/Android: Limit what apps can run via store
↳ also isolate

- To get on a standard iPhone, app has to get through review by Apple.
  ↳ Some limited checks for malware
  ↳ Also checks for biz reasons... in-app payments.
  * (can't have app offering loan for APR > 36% with repayment required ≤ 60 days, ...)
  * Epic suit

# iOS App Security

- Once app is on the phone, it runs in an isolated sandbox
  - * No shared files
  - * Only communication via limited APIs (photos)
- App developer can request access to extra APIs when submitting to app store
  - * Control VPN config
  - * Query user's location on push notification
  - * Get health data
  - ⋮

↳ Apple can try to limit API access to min necessary when app admitted to app store.

- Arguably these protections make it more difficult to get malware onto app store.

# Still, what can go wrong?

At start of semester, Nickolai mentioned XCodeghost

↳ Malicious version of XCode dev tools posted on pub mirror in China
(faster to download than true version in U.S.)

When developers compiled with malicious XCode, app did ???

↳ App store review didn't catch this
(though made it easier to clean up)

If isolation is so good, why care?

- Can get UUID of device, lang, country
- R/W clipboard (password manager, CC #, ...)
  ↳ could be very bad
- Can open URL — try to phish user (?)

↳ Could probably have been much worse (infected sensitive app)

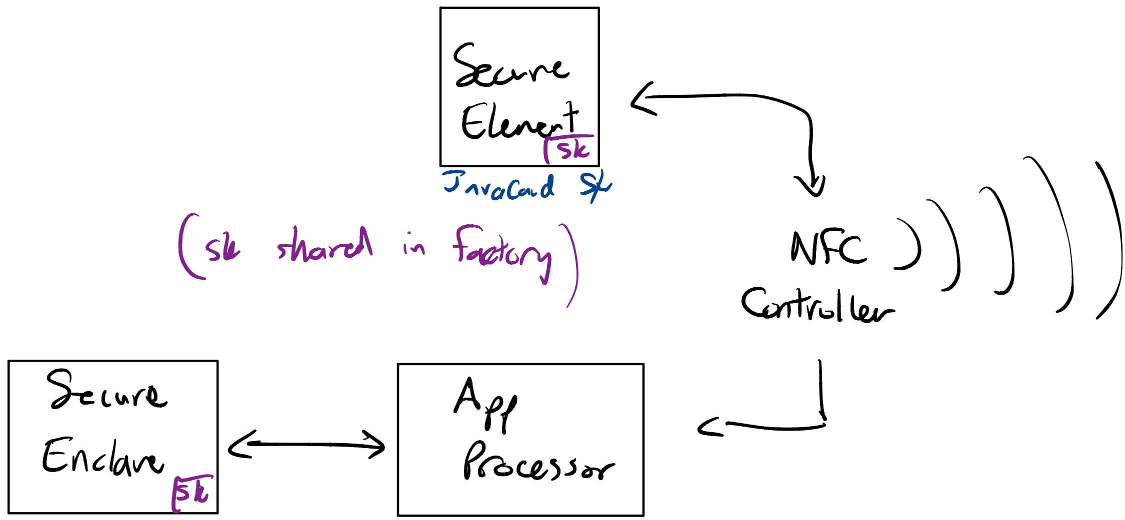
Can also steal user data more directly

- Fake Tor, ...

⇨ Still, isolation buys a lot.

# Buy stuff: Apple Pay

* CC data stored in secure "element" (not enclave)
  ↳ special payment applet that runs




* CC #s never stored on phone - device ID # provisioned when adding card to Apple Pay
  ↳ shared b/w phone and bank

* NFC payment data never touches App processor

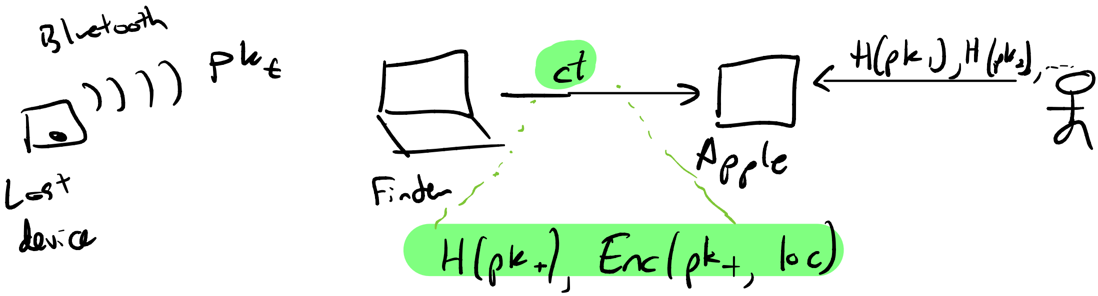
Purchase

1. User authenticates w/ passcode / touch ID (enclave)
2. Enclave sends req to element w/ special auth code fixed at ApplePay enable (deletable when wipe phone)
3. Element handles NFC txn with terminal
   ↳ element MACs txn data using key shared with payment network

# Leave Phone at Restaurant: Find my Phone

Can find offline devices (!)

↳ easy if online – device has GPS




To prevent pk from being a persistent tracking beacon, phone rotates pk

$$\begin{array}{ccccccc} sk_1 & \xrightarrow{H} & sk_2 & \longrightarrow & sk_3 & \longrightarrow & sk_4 \\ \downarrow & & \downarrow & & \downarrow & & \downarrow \\ pk_1 & & pk_2 & & pk_3 & & pk_4 \end{array}$$

→ Anyone can use this network to relay msgs (with AppleID)

**Phone stolen:** Remote wipe

E.g.,

- Phone grabbed in checked luggage
- Left in restaurant
- Taken by friend/partner

**Remote wipe:**

* App prc instructs enclave to erase secrets
  ↳ metadata probably left around but data
    & payments not possible

## Recap:

- Sophisticated & expensive defenses to protect against seemingly esoteric threats.
- Isolation and crypto combined at many layers
  - ↳ One not so good w/o the other.
- Raises bar for attack.