Client Device Security ①

if otherwise users → mobile single users

Design to protect your (sensitive) data if your (i) phone is stolen

- People keep secrets on business phones (e.g., government, reporters, medi)
- People nervous about passwords, personal info
- Competitive advantage against Android
- Do the right thing!

Assumptions

Device is passcode protected
Locked at time of theft

Potential Attacks?

Exhaustive search for passcode
(4 or 6 digits)

Impersonate user's face or fprint

Take apart phone & remove FLASH
- read from powered up RAM

Exploit bug in OS kernel
- install hacked version of OS
by attacking update server or
writing FLASH

↓ USB, WiFi active

↘ downgrade attacks

How successful?

FBI complains about difficulty

Backup your phone to iCloud government can get it from Apple. ECPA (1986) Stored Communications Act.

WhatsApp messages can be got from receiver (or sender) voluntarily or with warrent

high profile "breaks"

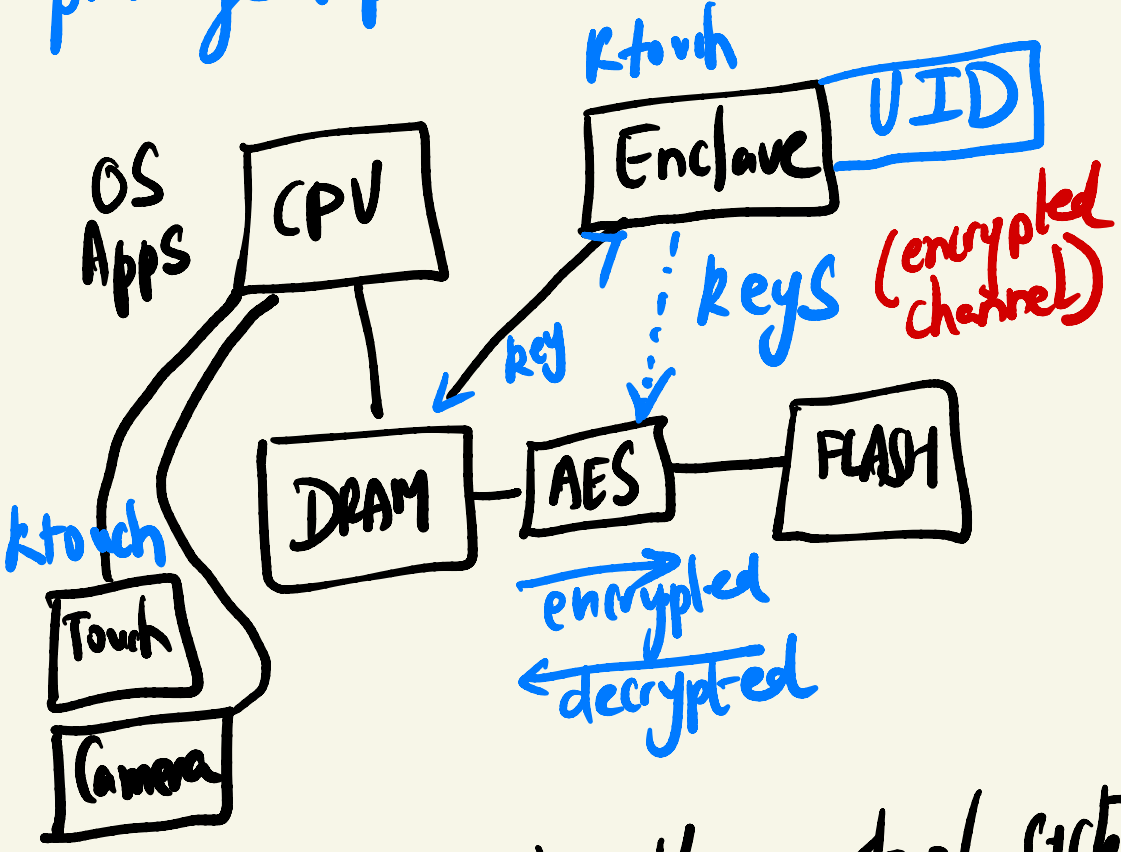
1000's of law enforcement agencies have tools to get into encrypted smartphones

for $15K a phone

— Grayshift, Cellebrite tools

iOS Hardware Architecture ④

privilege separation at hardware level



CPU least trustworthy part of system

Enclave has part of DRAM exclusive to it

encr. auth freshness

no hypervisor, small TCB.

Design focus

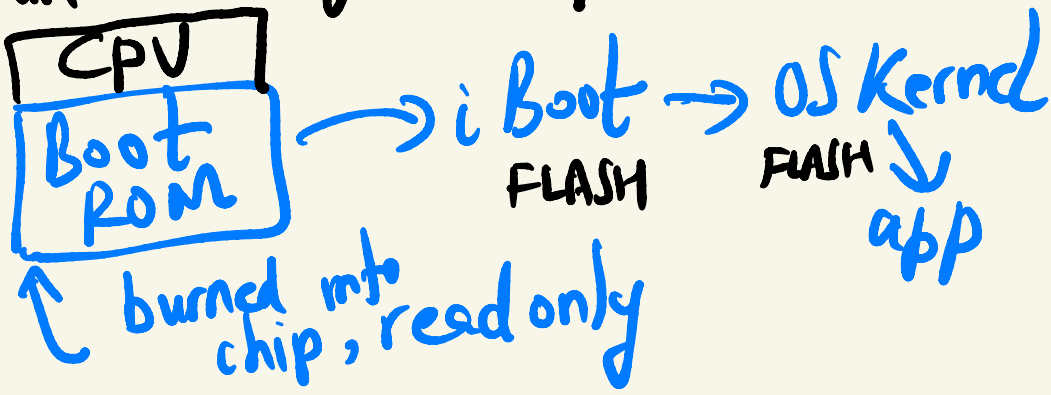
Secure boot

Enclave: hiding keys
data encryption gated
by passcode

Data encryption: running
background apps

Secure Boot

Make sure adversary cannot
run modified OS/apps/enclave



Public Key Signatures

Apple secret key SK_{apple}

PK_{apple} : public key

$Sign(SK_{apple}, m) = sig$

Device

$Verify(PK_{apple}, m, sig)$?

Boot ROM

PK_{apple}

read iBoot code from FLASH which has sig

$Verify(PK_{apple}, iBootcode, sig)$

and so on!

Secure Boot (contd.)

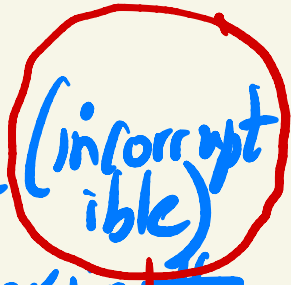
makes sure phone doesn't run arbitrary/unsigned code

Downgrade attacks

Known bugs in iBoot/kernel that are signed!

Boot the buggy kernel & exploit bug(s).

Strawman: Special storage (incorruptible) to store latest version #



Apple has different plan.

non-volatile memory

CPU can write but not adversary

Downgrade prevention

Specialize signature to a specific device

ECID: not secret, but different for every device (not easy to find)

CPU ECID

BootROM

Verify (iBoot || ECID, sig)

stateless Device

Apple

upgrade

ECID$_d$

check if latest version? If so

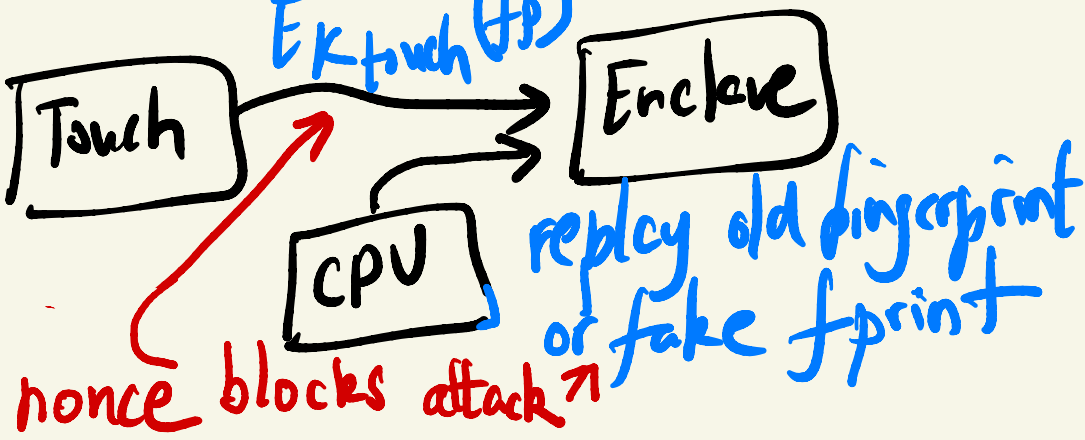
SK$_{apple}$ (iBoot, ECID$_d$)

'd' from huge space

Secure Enclave

- Prevent main CPU from seeing crypto keys that encrypt data
- Defend against passcode guessing
- Secure boot (like CPU), exclusive DRAM region which is encrypted, authenticated, checks freshness, using Merkle Tree.
- authenticate sensor I/O.

(compromised OS can't bypass checks)



Data encryption

FLASH data encrypted.
If phone is locked, need passcode to generate key to unlock/decrypt data.
(encryption key NOT on phone)

- Enclave h/w contains secret UID
- Enclave s/w cannot access UID but can only use to enc/dec
- $K_{PIN} = E_{UID}(E_{UID}(\cdots(passcode)))$ — 80ms
- Slow, enclave limits # guesses
- passcode not stored after reboot
- UID dependence ⇒ no bruteforcing from outside

Effaceable Storage

To avoid FLASH control, which makes data hard to erase.

Effaceable storage → raw FLASH

Enclave can reliably erase the keys stored on raw FLASH.

Weaknesses

UID extractable from hardware?
Apple's private keys must be kept secret.
USB, WiFi probably have bugs, active when phone is locked
Boot code/kernel may have bugs

Cost

Specialized hardware (enclave, FP sensor, AES DMA)
Passcode is annoying; forgotten passcode, data is lost if not backed up.
Background activities awkward