Mobile Device Security

- Design assumes:
  - Someone steals your iPhone
  - Device is passcode-protected and locked at time of theft
- Potential attacks:
  - Exhaustive search for passcode
  - Impersonate fingerprint / face
  - Take apart phones and remove flash storage / read from RAM
  - Exploit a bug in the OS kernel
    - USB / WiFi / radio might have bugs to exploit
  - Install a hacked version of the OS w/o security checks
    - I.e. redirect network traffic to use adversary's update server
    - Take apart phone and write different OS to flash chip

iOS Hardware Architecture

- Components
  - Main CPU
  - DRAM
  - Flash file storage
  - AES encryption engine between main CPU / DRAM and flash storage
  - Secure enclave CPU
    - UID cryptographic key in enclave hardware
    - We can use this to decrypt / encrypt but we cannot get this back out
  - Authentication sensors: fingerprint reader / face ID camera

Secure Boot

- Goal: make sure adversary cannot run modified OS / apps / enclave
- At power on, CPU starts executing from boot ROM that is read-only
- Boot sequence: boot ROM -> iBoot -> OS kernel -> apps
  - Boot ROM: checks iBoot code is signed by Apple's secret key
    - Checks it using Apple's public key
  - iBoot: checks OS kernel code is signed by Apple
  - OS kernel: checks apps are signed by developer
- How to check things are signed:
  - Apple holds some private key only known to them and code is signed
    - Code is hashed
    - This hash is then encrypted with Apple's private key
  - This hash is then decrypted using the public key stored in the secure enclave, then hash of the entire code has to match this decrypteed hash

Downgrade Attacks

- Downgrade attacks:
  - Replace OS kernel with an older one with a known bug
  - Bad solution:
    - iBoot checks OS kernel version is at least a certain version
    - Adversary could modify the stored version number
  - Good solution:
    - Each device has a unique ID called the ECID
    - Apple update servers signs each upgrade with a specific ECID

        - Boot sequence checks signature is for this phone's ECID
        - Good because:
            - Apple update servers will not sign an out-of-date kernel
            - Adversary unlikely to have old software with victim's exact ECID

Secure Enclave

- Purpose: prevent main CPU from ever seeing crypto keys
    - Also defend against passcode guessing
- CPU asks enclave to do things i.e. help decrypt certain things
    - Enclave is a separate CPU with a secure boot sequence
- Shares DRAM with the main CPU
    - Encrypts its own memory contents
- Runs fixed Apple software

- Overall plan: all user data is encrypted
    - Allow decryption only phone is unlocked
- Offloaded to separate CPU to prevent compromised OS kernel from getting decryption keys / fingerprint / face data or bypassing passcode retry limits
    - Allows hiding of keys with memory encryption / auth so direct read of RAM doesn't reveal keys
    - The main CPU cannot have all memory encrypted, but secure enclave is fine
- The decryption keys are computed based on passcode
    - More on this later, what ends up happening is only three or so keys are computed based on this and are used to wrap other keys
    - This is cached
        - Fingerprint / face recognition data can be used to retrieve cached value
    - We avoid having the OS communicate the sensor data directly to the secure enclave
        - Instead the sensor data is encrypted with manufacturing time secret key that the secure enclave knows

Data Encryption

- Files are encrypted in flash storage
- They generate data encryption keys based on the passcode rather than storing it permanently
    - This way they exist nowhere on device after reboot and can be forgotten when device locks
    - The encryption key is based on both the UID (hardware) of the secure enclave and the password
- The key is generated slowly to limit speed of exhaustive search, and enclave can also limit attempts
    - Dependent on UID to avoid brute forcing outside of the device
- Background apps:
    - When the phone is locked, some apps have to be able to run and read/write files
    - OS kernel can generate a secret key for each of these apps and then "wrap" the key with it for just this file and give it
    - The original actual key is no longer actually stored and only this app should be able to figure out what the actual key they need is

---

- The entire filesystem metadata is encrypted with a key that is based on the UID so that even if you take this flash chip out and put it in a different phone, it cannot read
    - This is given out to the OS once the phone boots
- Each separate file is encrypted with a different key, which is then wrapped by another key called the kdf before being sent back to the OS
    - Are stored in the secure enclave only
    - These keys can then be allocated out to different parts of the OS for different protection levels that are based on when the OS deletes the keys from its memory
        - Complete: can decrypt only when phone is currently unlocked

        - Derived from passcode on unlock, discarded when locked
    - Complete unless open: file can be written at any time but not read, used specifically for downloading attachments
    - Until first authentication
        - When phone was unlocked for the first time, this is derived
        - Discarded only on power-off
        - Default for third-party app data
    - No protection: can decrypt any time: only derived from UID and not passcode
  - After the OS deletes the key, it will have to go back to the secured enclave with the password and re-ask for the key, which will require the password
- To use these wrapped file keys:
  - FS code on CPU asks secure enclave to get certain keys for certain files
  - CPU never receives raw keys, but instead wrapped ones
  - The secure enclave and the AES engine (between the CPU and its flash memory) share a secret
    - The CPU then has to send this to the AES engine to "program" it, so the CPU still never sees the actual key
- This allows you to change the pin fast:
  - You just need to recompute these kdf keys (only 3-4) and then you're good because the actual file keys don't have to change
  - Secure enclave will then just have to start encrypting the filesystem keys with new stuff