# iOS Security Overview

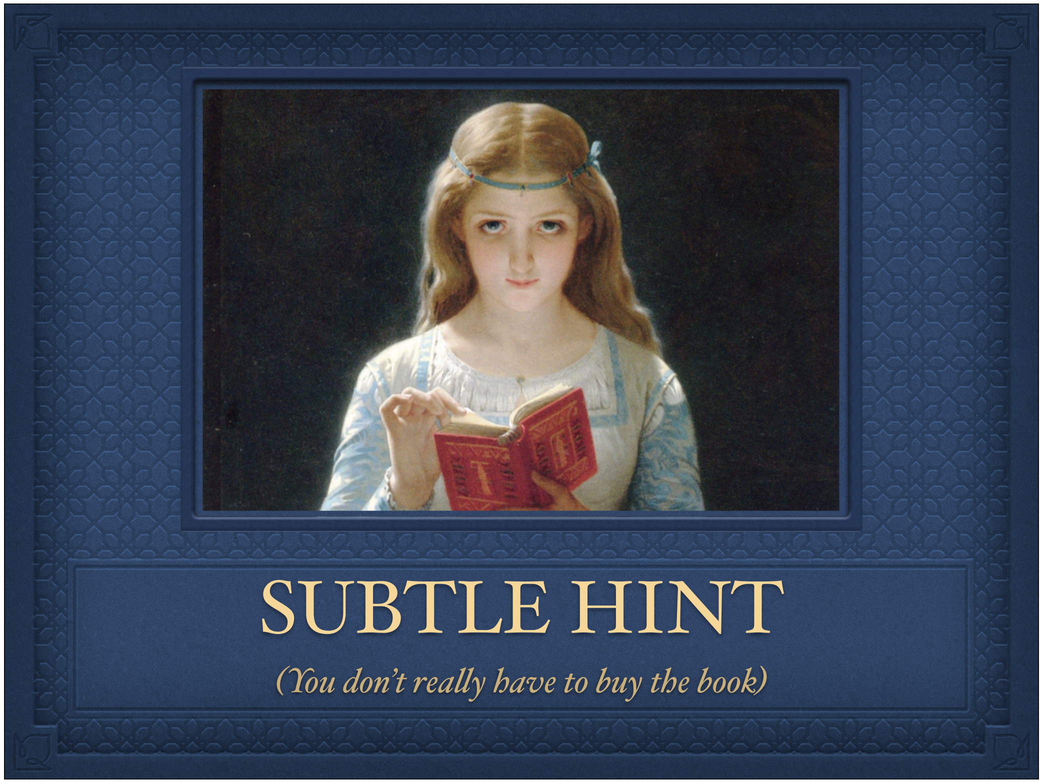
*Allister Banks for the July 2015 Philly Mac Admins Meetup*

# iOS Security Overview

*Allister Banks for the July 2015 Philly Mac Admins Meetup*

As it said on the meetup invite my name is Allister Banks, and for a change of pace I'll be discussing iOS Security. I work for a company called Montefiore Health Systems, and our IT division is considered a peer to the 5-ish and growing hospitals we run. My boss had the stat that we account for one of the most emergency room trips in the country if not the world… I'm the technical lead of the Apple Support Team, and we care for 4 digit iOS devices which I expect will go to 5 over the next year or two, but only 3-digit macs at this point.

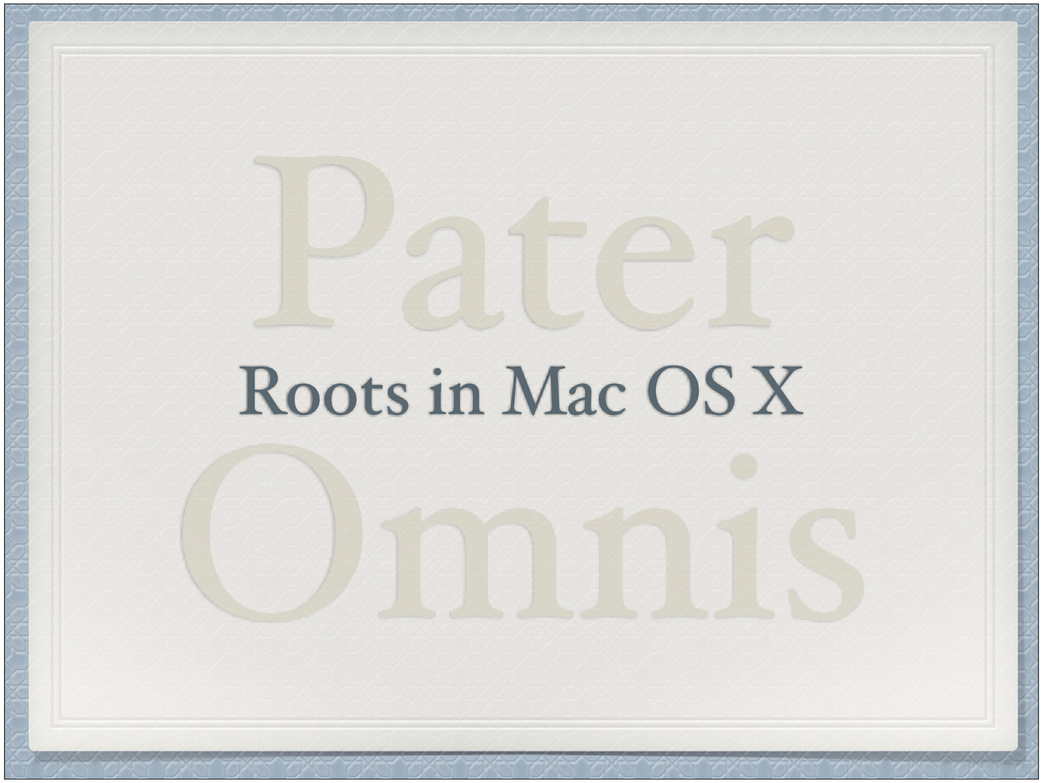
You may know me from such films as... I mean, I think I'm primarily known in the community for working with Macs, but I previously ran trainings for Apple's VAR program on their iOS solutions and had the extreme honor and pleasure of writing a small book...



SUBTLE HINT

*(You don't really have to buy the book)*

with a guy named Charles Edge for Packt Publishing, which I'm not saying anyone has to run out and buy,
but I think it turned out pretty well. Anyway, this is considered a Mac Admin meetup, but I'm going to hopefully hip y'all with an overview of the security-specific features of iOS, and show off some of my notes that helped me understand the topic a little better, although you admittedly could just reading their 55 page whitepaper on the topic. It’s certainly as authoritative as you get.



Pater

## Roots in Mac OS X

Omnis

Getting into it, as you all should know, they did not go with their linux variant that had been in use for old iPods when they designed the iPhone, and now with the WatchOS, OS X can now be loosely considered the father of them all, they all evolved from that same Kernel. But experience-wise the similarities pretty much end there, although command-tab app switching and other keyboard and productivity enhancements in iOS9 are the rare instance of bringing Mac to iOS. In sheer volume of unit sales, tho, you can tell which side of their bread is buttered, innovation-wise.

# Bootchain through Apps

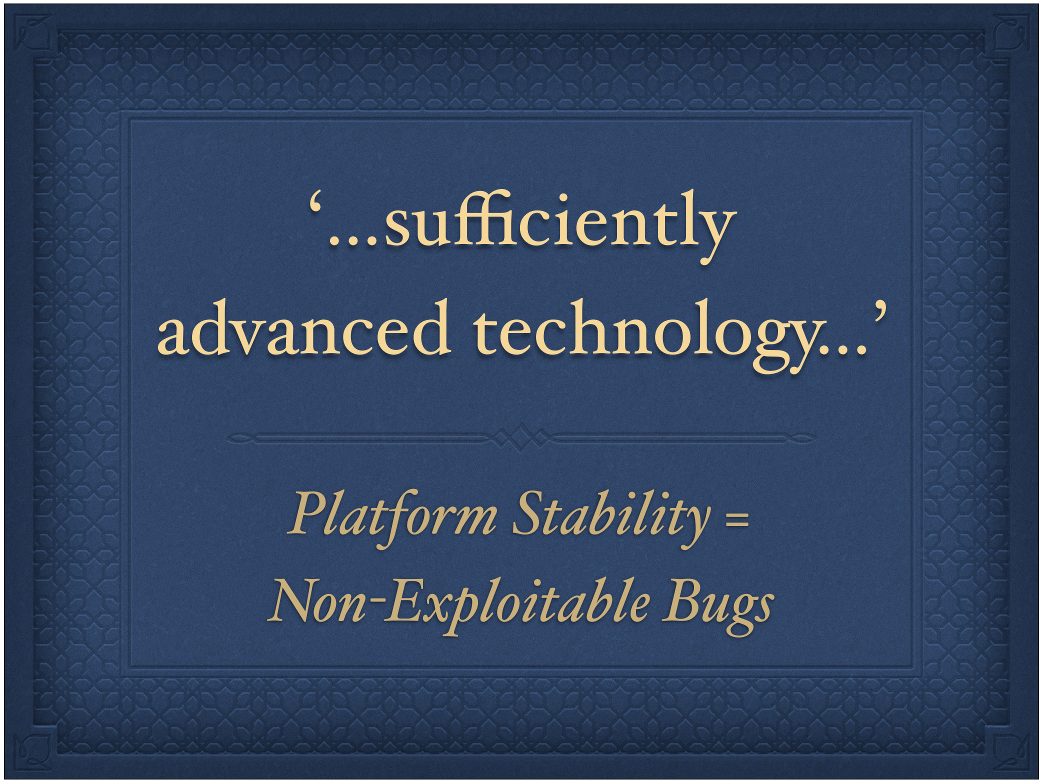
*(Apologies to Gaugin)*

For the macadmin crowd, I've geared this to be the same kind of training you probably encountered when studying for your first OS X certification test, and that is you're expected to know how to debug or troubleshoot the boot process, by becoming aware of its elements or moving parts. So no listing restrictions, no tools you'd use when interacting with managing the device, so no MDMs, just the security architecture. It's dry, so I'm going to keep up a good pace and speed through it

# Why the Walled Garden?

*Platform Stability = Non-Exploitable Bugs*

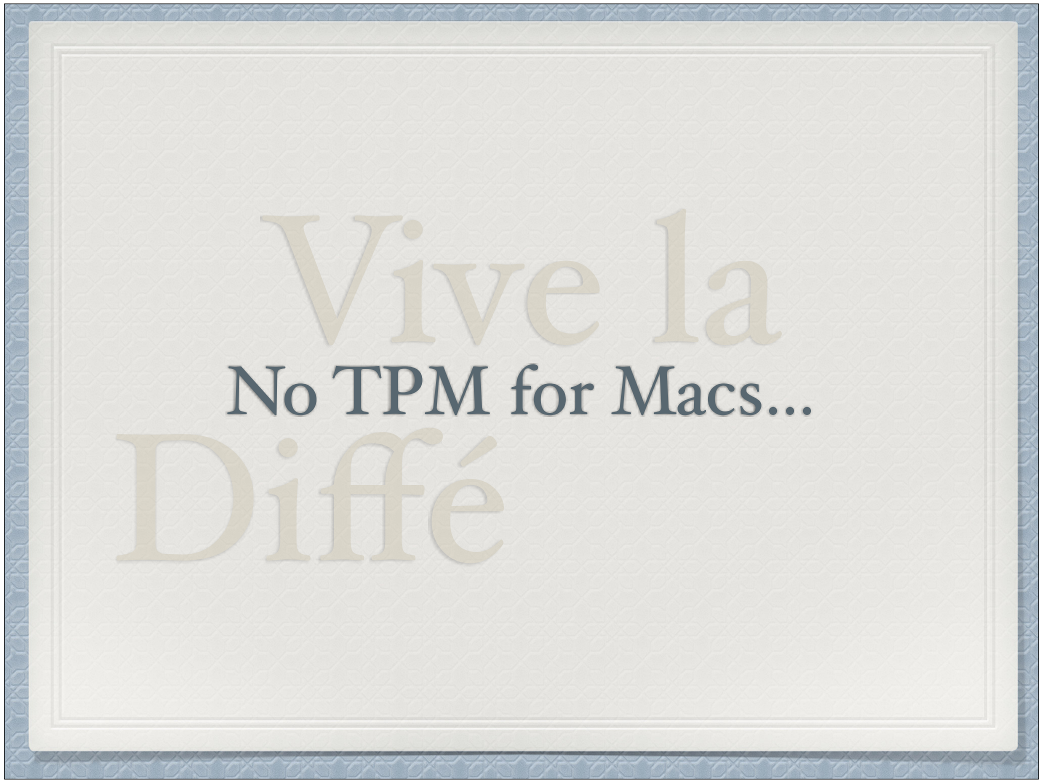
Why they started out on a path to have iOS be head-and-shoulders above all other commercially viable platforms available (IMHO) when it comes to security is a good question, and it’s hard to put faith in their review process as the only thing holding the barbarians at the gate, which we’ll touch on later.



‘...sufficiently advanced technology...’

*Platform Stability = Non-Exploitable Bugs*

And as you may recall from the first ways people ran 3rd party apps on the phone before the existence of the App store, they exploited bugs - you call them jailbreaks, I call them exploits. So for Apple wanting to sell you apps or control access to them, their business incentive to have a secure platform may have helped them focus their priorities. Apologies to Arthur C Clarke's three laws, but a sufficiently laxidasical security stance is indistinguishable from a bug-ridden crappy co-opted experience.



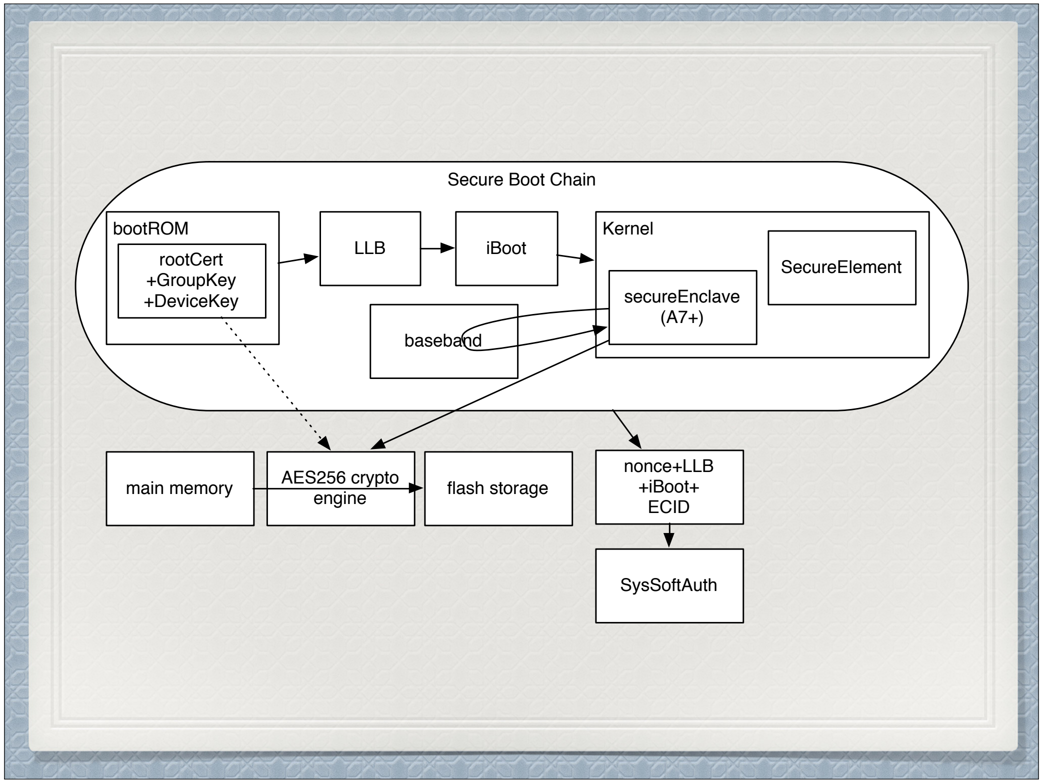
Vive la

## No TPM for Macs…

Diffé

Us Mac Admins know this is not anything like the way it is on OS Ten, most noticeably when it comes to the lack of utilizing Intels Trusted Protection Module and therefore exploits like Thunderstrike are real and really egregious.

Anyways, now we're going to get into my cliffnotes lecture-like summation of Apple's security whitepaper with a bunch of graffles I put together, with some exposition on the more interesting points or barriers to understanding that I overcame during the process of writing the book.



Secure Boot Chain

bootROM

rootCert
+GroupKey
+DeviceKey

LLB

iBoot

Kernel

SecureElement

secureEnclave
(A7+)

baseband

main memory

AES256 crypto
engine

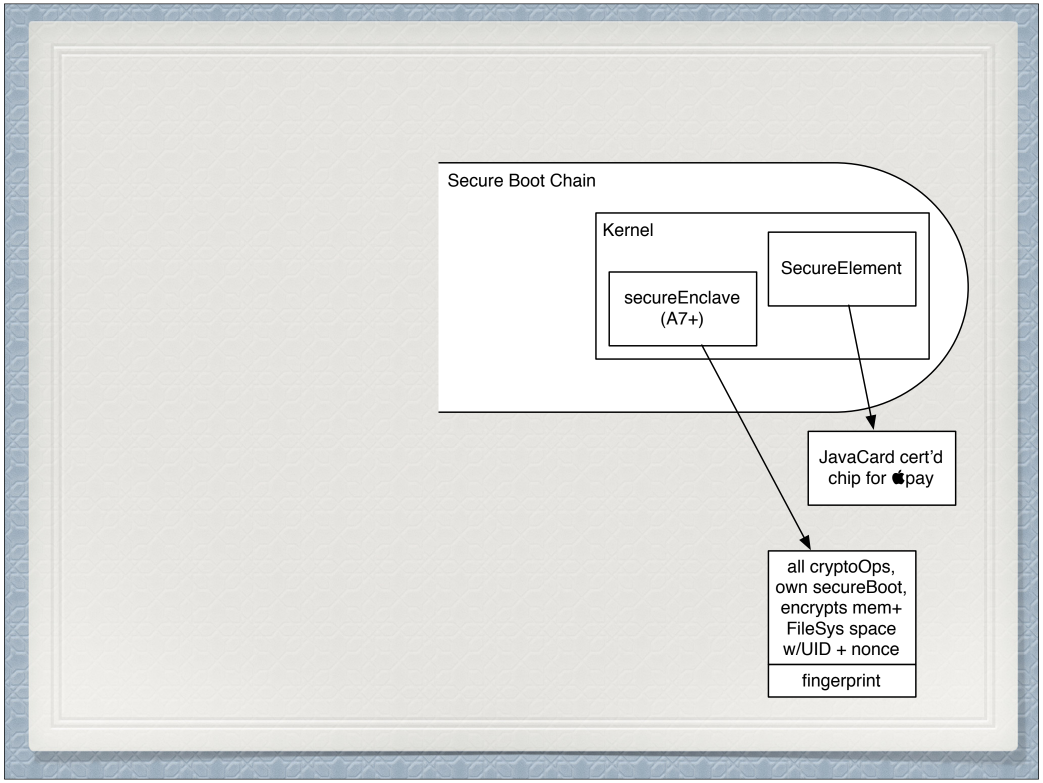
flash storage

nonce+LLB
+iBoot+
ECID

SysSoftAuth

Launchd is still process zero, but we have a chain of trust from firmware where it jumps several hurdles in order to validate that things haven't been tampered with or are otherwise non-functional.

- We start with the private keys needed for the device to have its own identity and that model of device to have a group key, both of which are literally burned in at the factory. Apple's root CA cert is there, and for the tin-foil-hat crowd out there, this process is not performed without Apple QA employees being present. I could get deeper into that but let me gloss over it for now, suffice it to say it's the ingedient that's the basis of truth for all cryptographic operations henceforth, so a privacy-concerned apple find that to be a critical juncture.
- Then the low-level-bootloader gets the handoff from read-only firmware on the device and in turn makes sure that the iOS-specific iBoot process about to take over is signed and secure, otherwise if there's any failure with either side you're going to get ye olde connect to iTunes icon.
- If a new OS version comes out and is applied to a device, the process that happens during iBoot also ensures you can't downgrade, which I'll explain further later, but these components build into a mechanism called SystemSoftware Authorization, you wouldn't be able to complete activation on a device that's checked in with apple as having the newer OS once.



Secure Boot Chain

Kernel

secureEnclave (A7+)

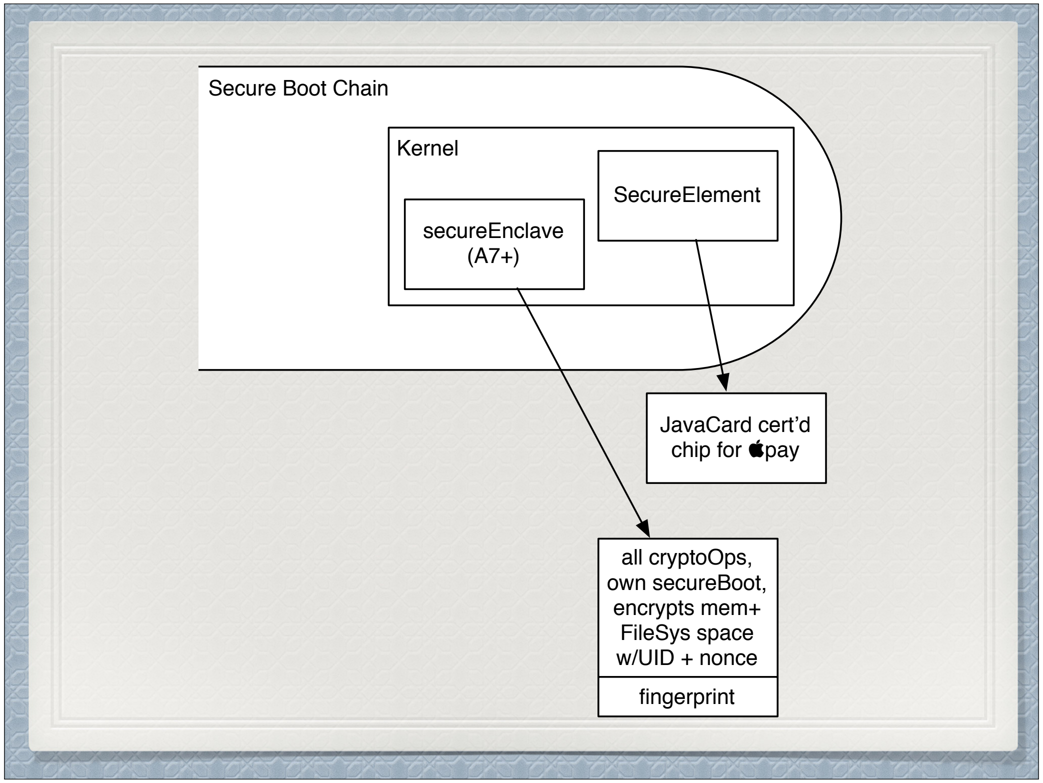
SecureElement

JavaCard cert'd chip for pay

all cryptoOps, own secureBoot, encrypts mem+ FileSys space w/UID + nonce

fingerprint

Now zooming in on more recent developments,



Secure Boot Chain

Kernel

secureEnclave (A7+)

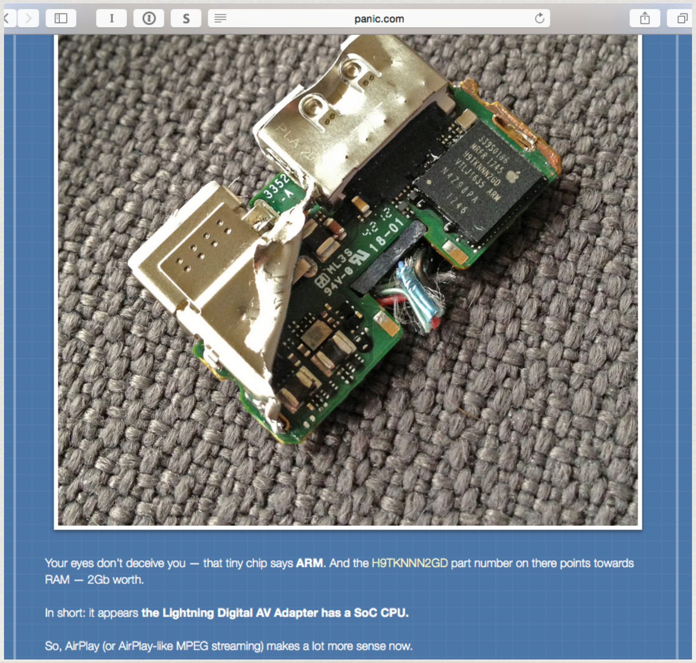
SecureElement

JavaCard cert'd chip for pay

all cryptoOps, own secureBoot, encrypts mem+ FileSys space w/UID + nonce

fingerprint

- As of the A7 and iPhone 5S we have the secure enclave for dedicated cryptographic processing, which is how things can go from being in-memory representations to encrypted on the storage without overhead to the main cpu. It also now verifies the cellular network baseband as well as it's own startup process, kept separate from rest of the system.
- The secure enclave is where fingerprints get sent, as it has it's own separate secured storage, and the payment standard apple uses was rebranded from JavaCard for understandable reasons. The same security applied to storing the fingerprint is on the payment cards added to apple pay.
- This is not so different an architecture from google Santa process checking kernel extension that they released, you only have a non-exploitable security measure if you protect the kernel layer, which is how Santa works.

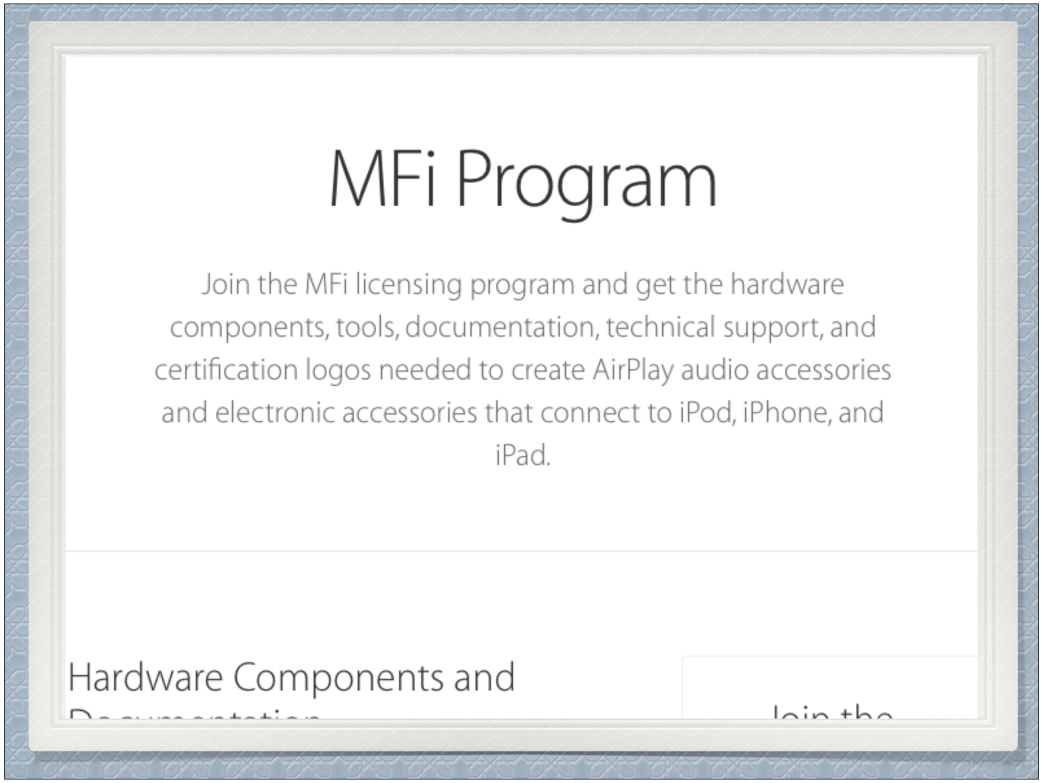


Your eyes don't deceive you — that tiny chip says **ARM**. And the H9TKNNN2GD part number on there points towards RAM — 2Gb worth.

In short: it appears **the Lightning Digital AV Adapter has a SoC CPU.**

So, AirPlay (or AirPlay-like MPEG streaming) makes a lot more sense now.

Ok, most of the big strokes of getting from the boot process through to iOS's springboard done.
Lightening things up a bit. You may recall the hdmi to lightning adapter actually has a circuit board in the casing that the device is in essence airplay'ing its videostream through, as reported on the panic blog. And i mention this to underline a fact you may not know,



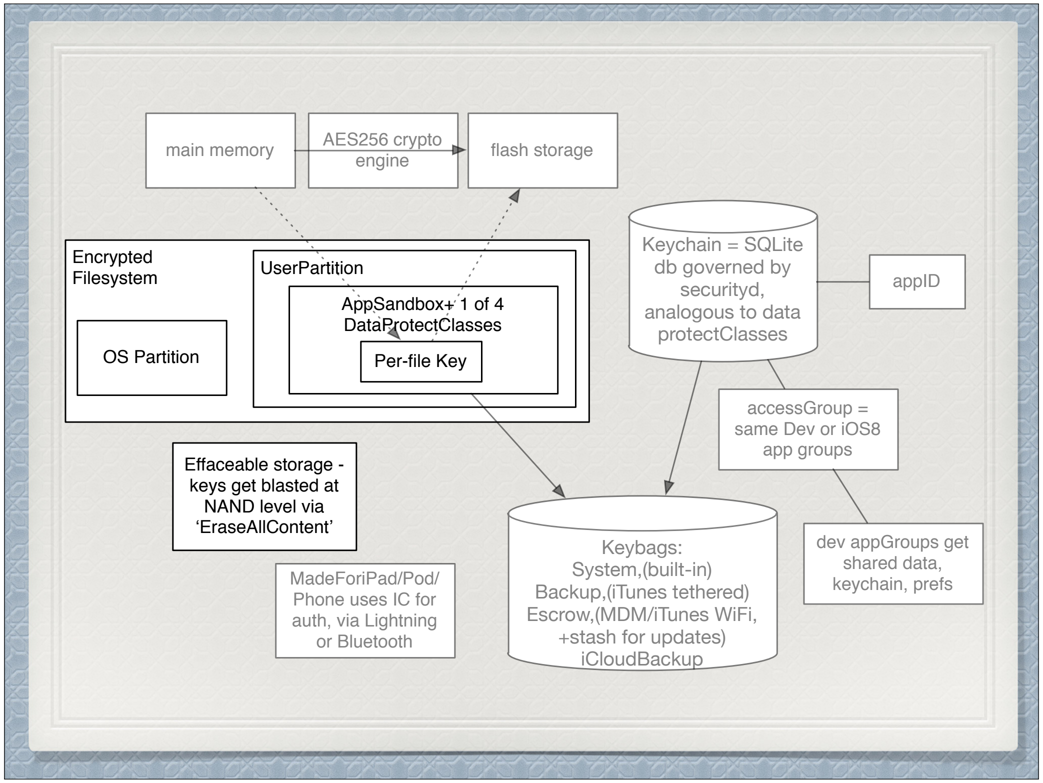
# MFi Program

Join the MFi licensing program and get the hardware components, tools, documentation, technical support, and certification logos needed to create AirPlay audio accessories and electronic accessories that connect to iPod, iPhone, and iPad.

Hardware Components and Documentation

Join the

all lightning and bluetooth devices that carry the Made for iPhone/iPad/iPod certification actually have auth via an integrated circuit, so there is some amount of assurance you're not getting pwn3d by the cable itself.



main memory

AES256 crypto engine

flash storage

Encrypted Filesystem

OS Partition

UserPartition

AppSandbox+ 1 of 4 DataProtectClasses

Per-file Key

Keychain = SQLite db governed by securityd, analogous to data protectClasses

appID

accessGroup = same Dev or iOS8 app groups

dev appGroups get shared data, keychain, prefs

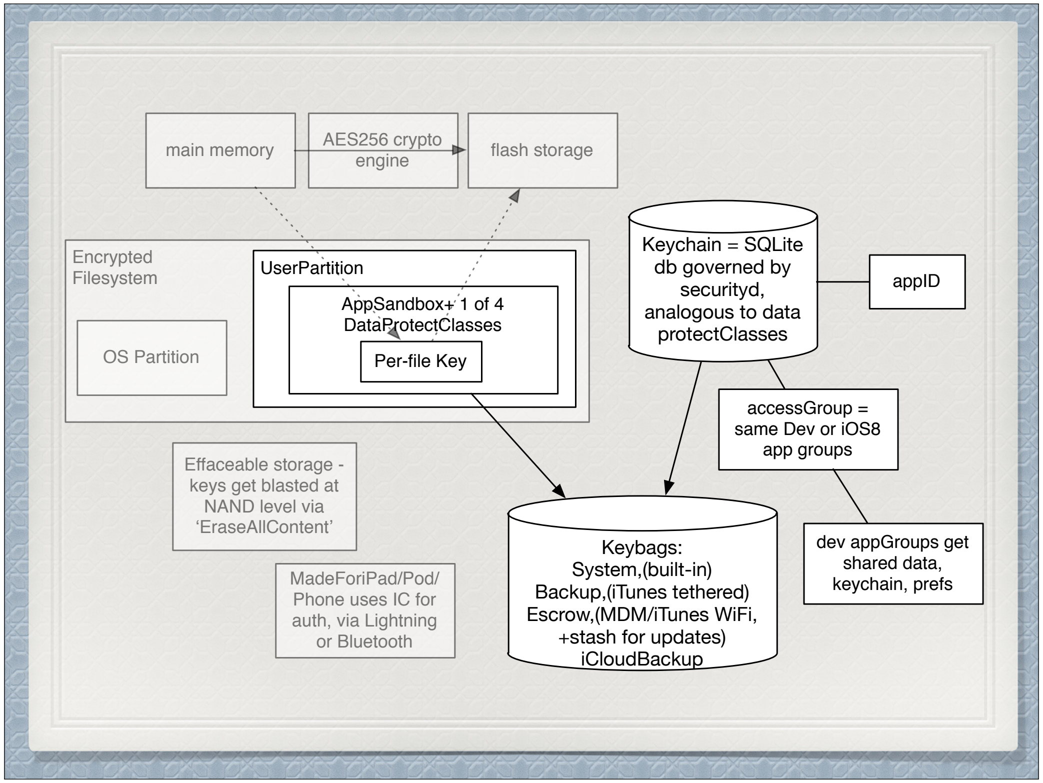
Effaceable storage - keys get blasted at NAND level via ‘EraseAllContent’

MadeForiPad/Pod/Phone uses IC for auth, via Lightning or Bluetooth

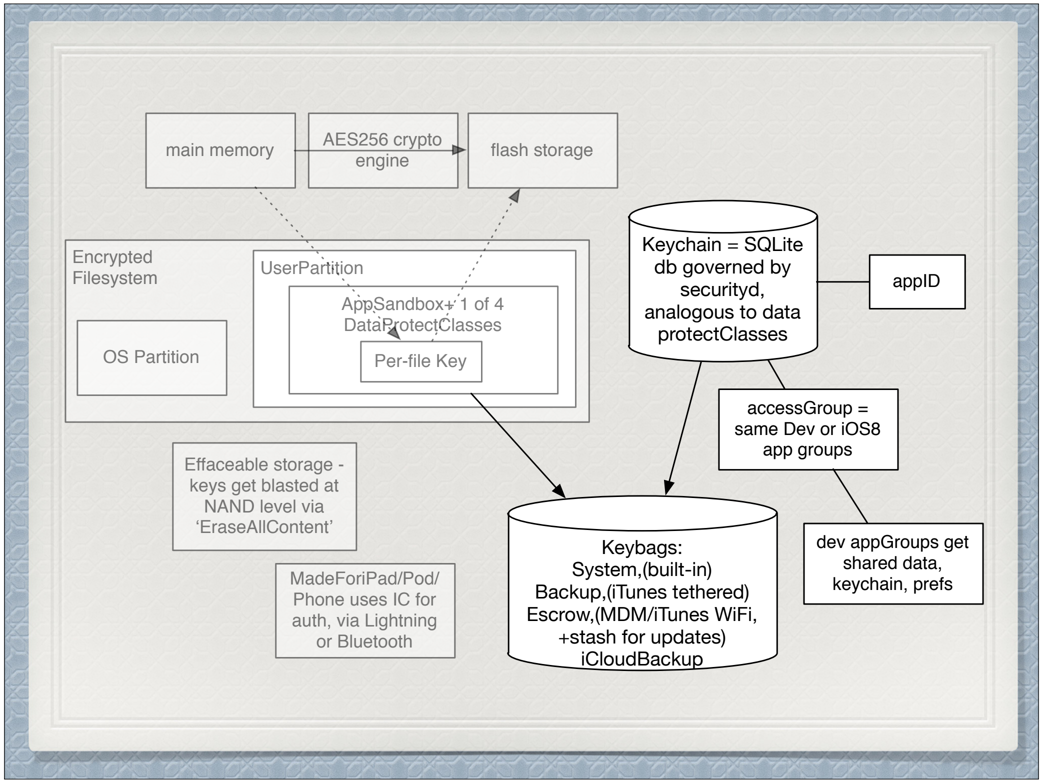
Keybags:
System,(built-in)
Backup,(iTunes tethered)
Escrow,(MDM/iTunes WiFi, +stash for updates)
iCloudBackup

I called out the secure enclave's CPU as being right in the pipeline between the data as it’s operated on in memory and the system storage, but when it gets involved in supporting security for the rest of just the OS could use some explaining. Pulling back to a more basic level, the storage is in itself encrypted, but rarely is the device shutdown for you to get the benefit of that protection. Previously a lot of the extent of the security applied was based on the presence of a passcode, and that's still important,

- but for those that weren’t aware, the entire storage volume since iPhone 3G has a key which is kept in a dedicated part of the NAND called effaceable storage so that the wear-leveling properties that flash has, which wouldn’t therefore provide secure erase, don't become a liabliity when you go to wipe the device.
- To begin to protect the OS itself after the whole boot verification process, it's on its own read-only partition, and so we should only need to be concerned with the user partition and how apps store data per-file.


main memory
AES256 crypto engine
flash storage
Encrypted Filesystem
OS Partition
UserPartition
AppSandbox+ 1 of 4 DataProtectClasses
Per-file Key
Keychain = SQLite db governed by securityd, analogous to data protectClasses
appID
accessGroup = same Dev or iOS8 app groups
dev appGroups get shared data, keychain, prefs
Effaceable storage - keys get blasted at NAND level via 'EraseAllContent'
MadeForiPad/Pod/Phone uses IC for auth, via Lightning or Bluetooth
Keybags: System,(built-in) Backup,(iTunes tethered) Escrow,(MDM/iTunes WiFi, +stash for updates) iCloudBackup

- There are four of what's called protection classes that govern when files are unlocked from encryption when the device is running, mainly having to do with if the passcode has been entered or if you've auth'd to an app, like using touchID for the 1Password app. That also explains why you get mail notifications without it actually downloading your mail, the built-in mail app stores its data encrypted and does not put more data on the device while it's locked because it doesn't have access to your encryption key until you unlock it.



main memory

AES256 crypto engine

flash storage

Encrypted Filesystem

OS Partition

UserPartition

AppSandbox+ 1 of 4 DataProtectClasses

Per-file Key

Keychain = SQLite db governed by securityd, analogous to data protectClasses

appID

accessGroup = same Dev or iOS8 app groups

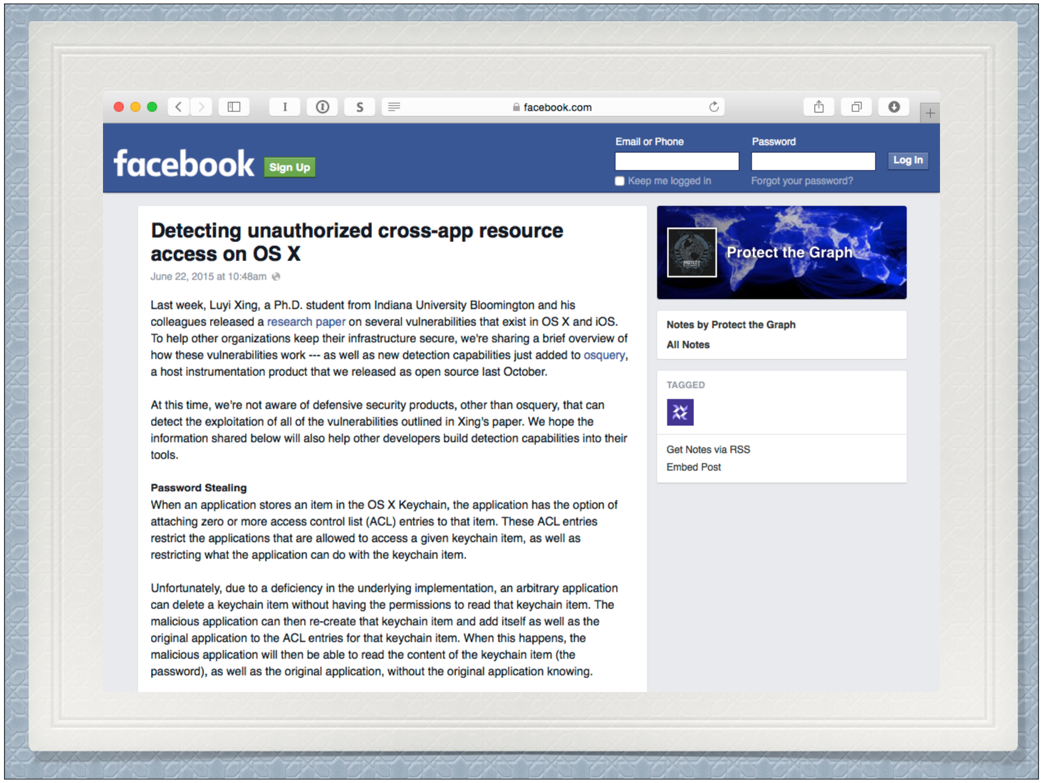
dev appGroups get shared data, keychain, prefs

Effaceable storage - keys get blasted at NAND level via ‘EraseAllContent’

MadeForiPad/Pod/Phone uses IC for auth, via Lightning or Bluetooth

Keybags:
System,(built-in)
Backup,(iTunes tethered)
Escrow,(MDM/iTunes WiFi,
+stash for updates)
iCloudBackup

- Moving along quickly, a keybag is a collection of keys, which can be bulk-unlocked for operations like sending a backup, and that concept allows a decoupling of the individual keys that are needed to keep those moving parts of needing to practically unlock everything secure, a la defense in depth.



## Detecting unauthorized cross-app resource access on OS X

June 22, 2015 at 10:48am

Last week, Luyi Xing, a Ph.D. student from Indiana University Bloomington and his colleagues released a research paper on several vulnerabilities that exist in OS X and iOS. To help other organizations keep their infrastructure secure, we're sharing a brief overview of how these vulnerabilities work --- as well as new detection capabilities just added to osquery, a host instrumentation product that we released as open source last October.

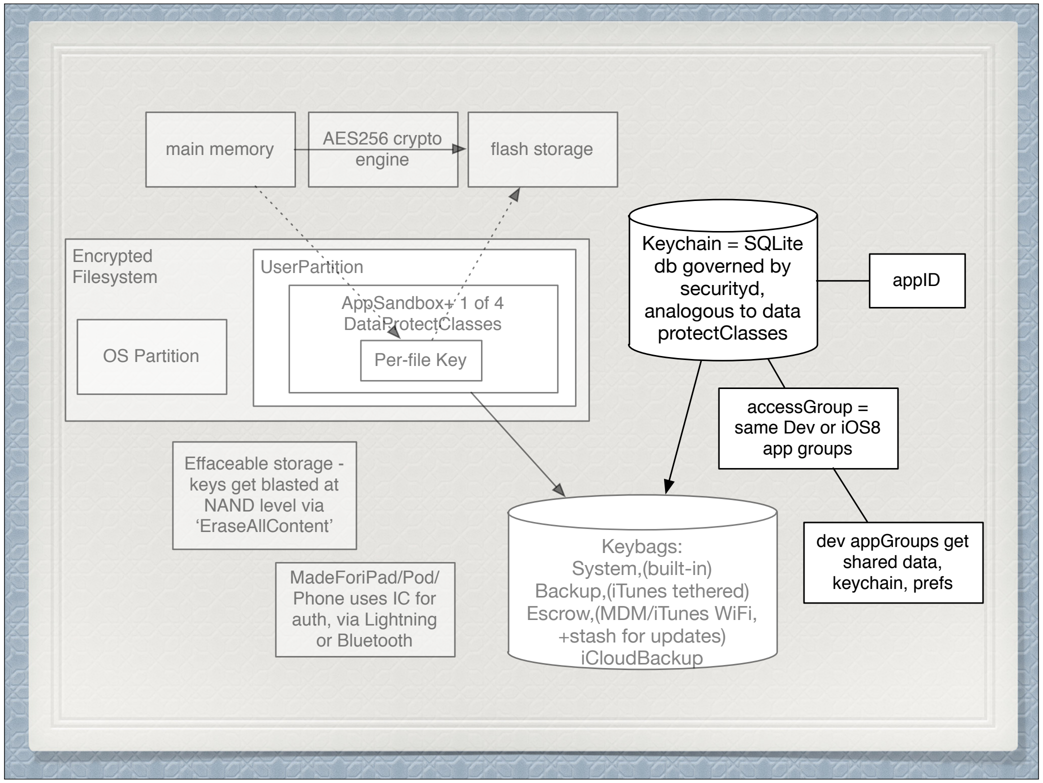
At this time, we're not aware of defensive security products, other than osquery, that can detect the exploitation of all of the vulnerabilities outlined in Xing's paper. We hope the information shared below will also help other developers build detection capabilities into their tools.

**Password Stealing**

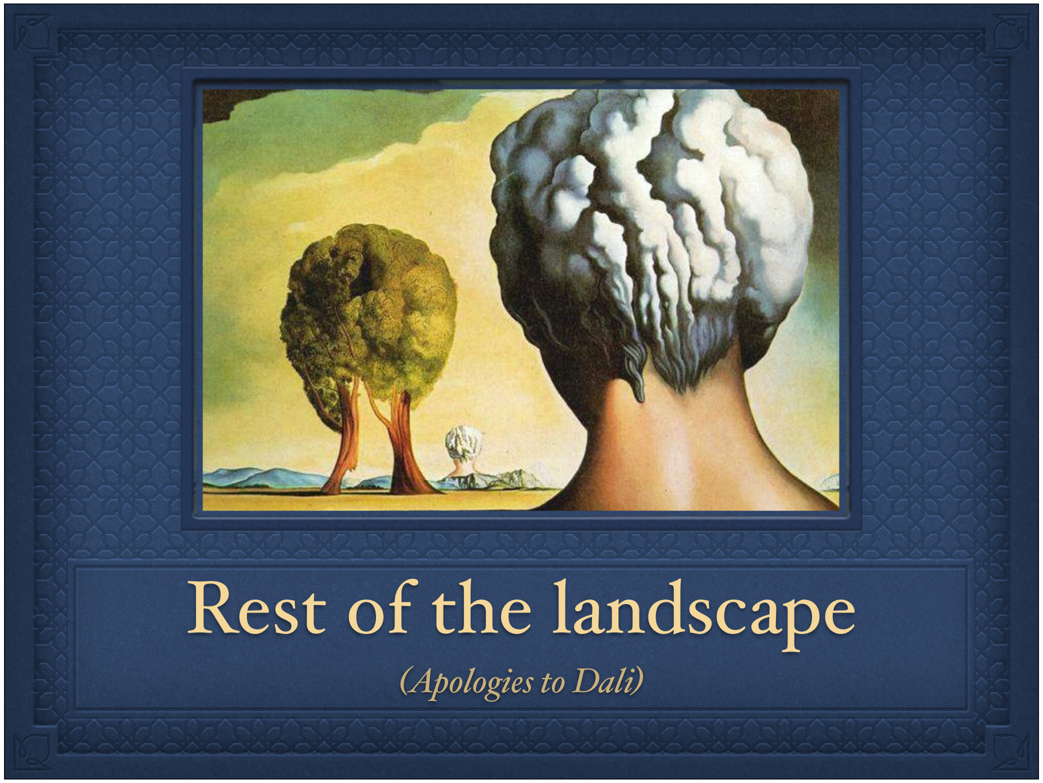
When an application stores an item in the OS X Keychain, the application has the option of attaching zero or more access control list (ACL) entries to that item. These ACL entries restrict the applications that are allowed to access a given keychain item, as well as restricting what the application can do with the keychain item.

Unfortunately, due to a deficiency in the underlying implementation, an arbitrary application can delete a keychain item without having the permissions to read that keychain item. The malicious application can then re-create that keychain item and add itself as well as the original application to the ACL entries for that keychain item. When this happens, the malicious application will then be able to read the content of the keychain item (the password), as well as the original application, without the original application knowing.

- And as came up recently on the Mac and iOS, security researchers have been looking at how credentials are passed between apps, with access control lists, just like on fileshares and network devices, being exploited by Apple's previously naïve stance at allowing incorrect parties to register with the system as trusted. And they allowed apps in the Mac and iOS app stores that exhibited bad behavior when it comes to getting secrets stored by other apps, so as I mentioned earlier the review process isn’t flawless.
- This is a post by the osquery team at facebook describing how they can look at a system behaviorally to see when the elements of this technique could be in use, so they don't need to fingerprint an attack vector to stop it or prevent it from spreading. That might have been a thick statement, so if you're interested please ask me more later.


main memory
AES256 crypto engine
flash storage
Encrypted Filesystem
OS Partition
UserPartition
AppSandbox+ 1 of 4 DataProtectClasses
Per-file Key
Keychain = SQLite db governed by securityd, analogous to data protectClasses
appID
accessGroup = same Dev or iOS8 app groups
dev appGroups get shared data, keychain, prefs
Effaceable storage - keys get blasted at NAND level via 'EraseAllContent'
MadeForiPad/Pod/Phone uses IC for auth, via Lightning or Bluetooth
Keybags: System,(built-in) Backup,(iTunes tethered) Escrow,(MDM/iTunes WiFi, +stash for updates) iCloudBackup

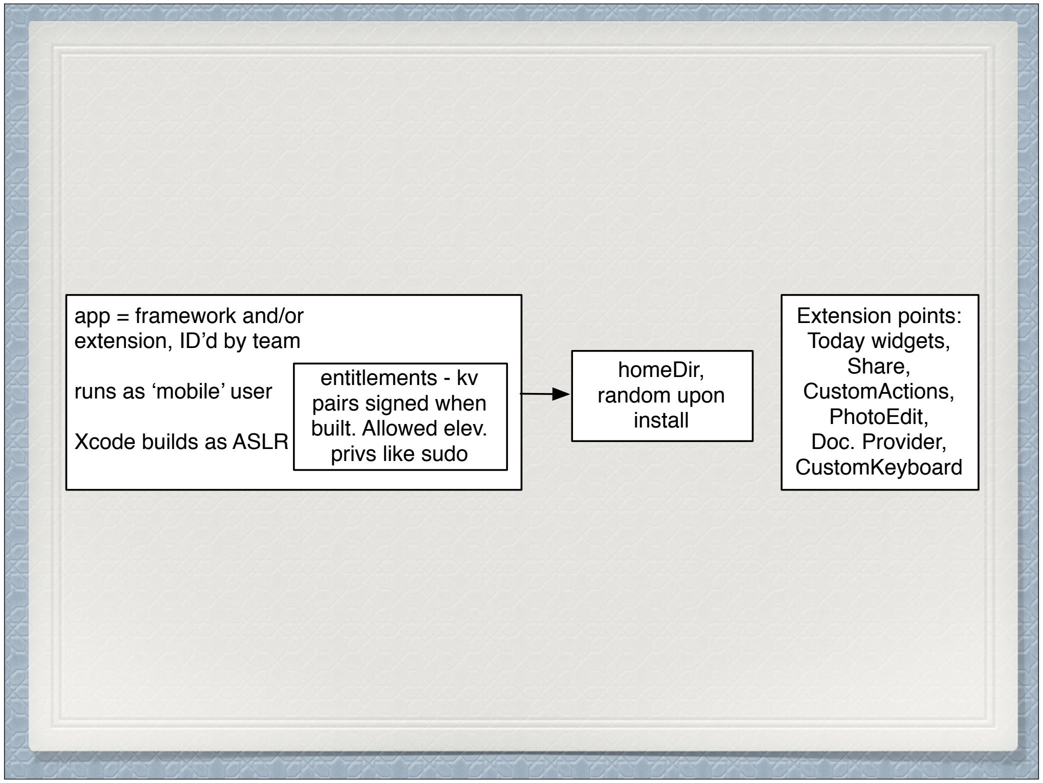
- so keybags are the macro level, individual keychain entries like literal rows in a database have classes applied to them along with access control, and that access is granted to apps by the same developer or explicitly allowed groups, like when there is dropbox or 1password integration.



# Rest of the landscape

*(Apologies to Dali)*

Everyone can stand up and take a stretch to keep your watches happy if you'd like, we're over 2/3rds through, so I'm just going to cover the rest of the basics, which includes some aspects of apps, networking in general and VPN in specific before we wrapup



app = framework and/or extension, ID’d by team

runs as ‘mobile’ user

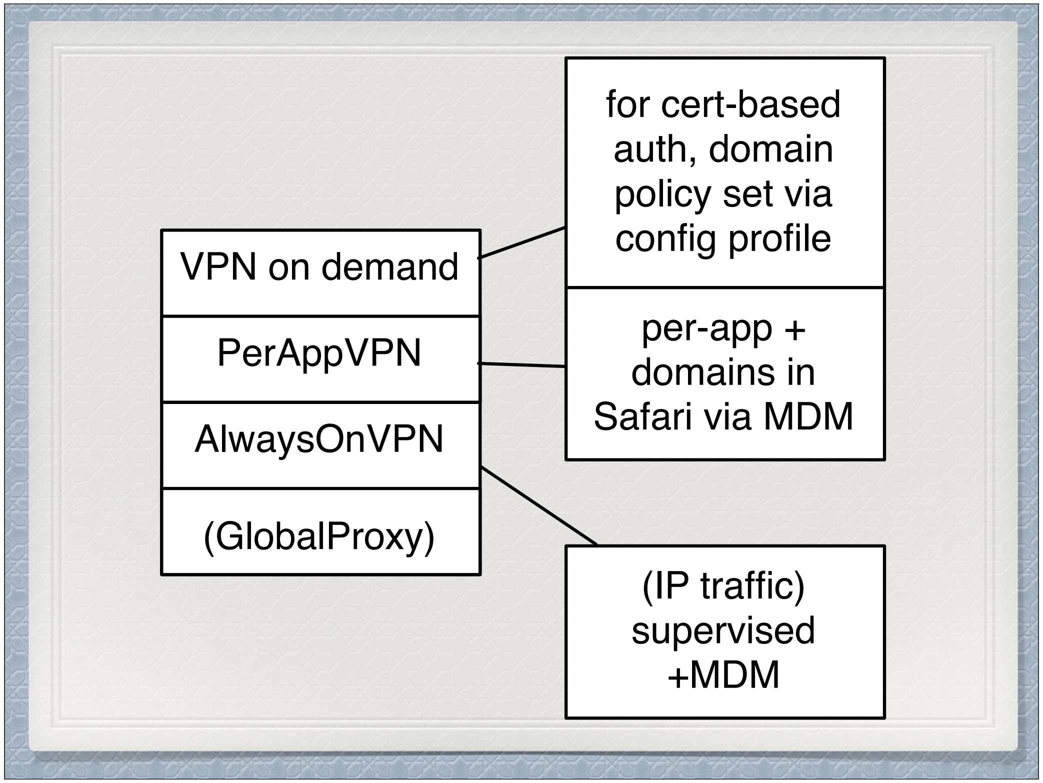
Xcode builds as ASLR

entitlements - kv pairs signed when built. Allowed elev. privs like sudo

homeDir, random upon install

Extension points: Today widgets, Share, CustomActions, PhotoEdit, Doc. Provider, CustomKeyboard

- For the last parallel between Mac and iOS, there's a symlink on the OS partition to the user named 'mobile's home folder on the other partition, and that's who all the apps run as. ‘This is a unix system, I know this!’ Each app, id'd by team or dev, can be granted background-type processes by stating the request for the entitlement in the info.plist for when an app is submitted to apple, which Xcode handles in addition to setting the app up to take advantage of address space layout randomization when built. In addition to that exploit mitigation, the home folder sandbox itself is given a random location on the user partition. And just as a reminder apps can poke their functionality into other apps through extensions, whihc are in these 6 categories only at this point. I'm really hoping they refine this so I don't go to a share button to use 1password, I'm a little surprised they haven't stated that they plan to push on that front for iOS9, but hey - it increases the shelflife of my book...



VPN on demand

PerAppVPN

AlwaysOnVPN

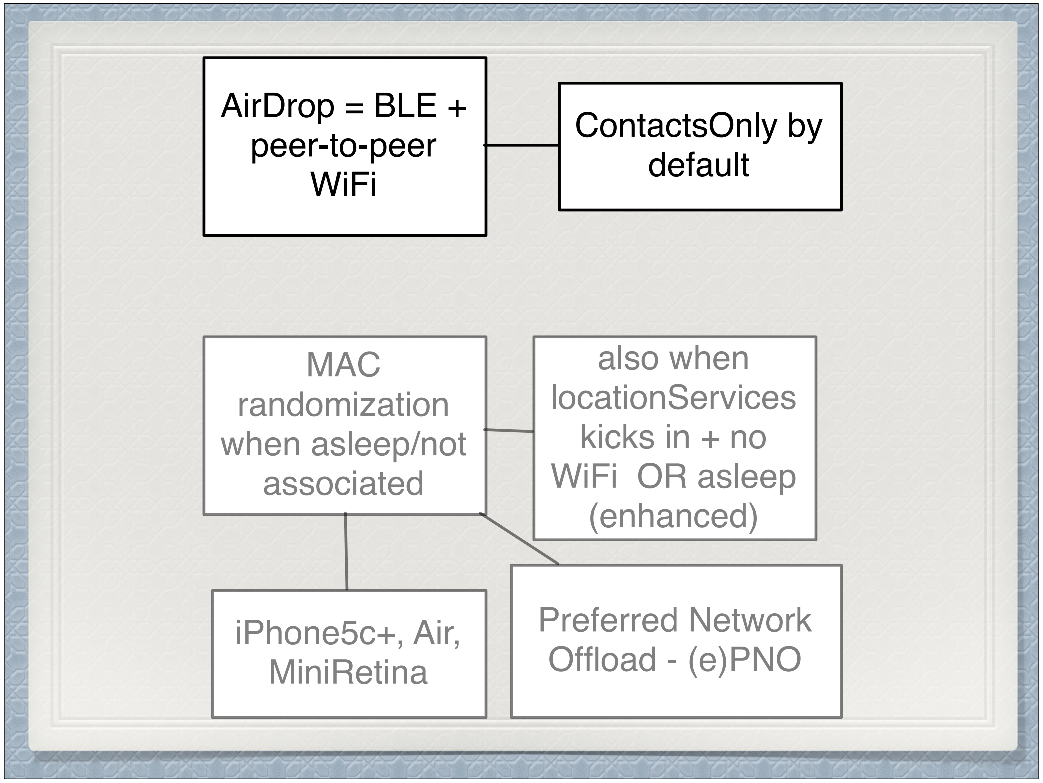
(GlobalProxy)

for cert-based auth, domain policy set via config profile

per-app + domains in Safari via MDM

(IP traffic) supervised +MDM

- Networking-wise these are the tiers of VPN or proxying options available, which Apple has stated they'll expand in iOS9 to domain black and whitelisting on the proxy level - if anyone's interested in any of these I can go into them...



AirDrop = BLE + peer-to-peer WiFi

ContactsOnly by default

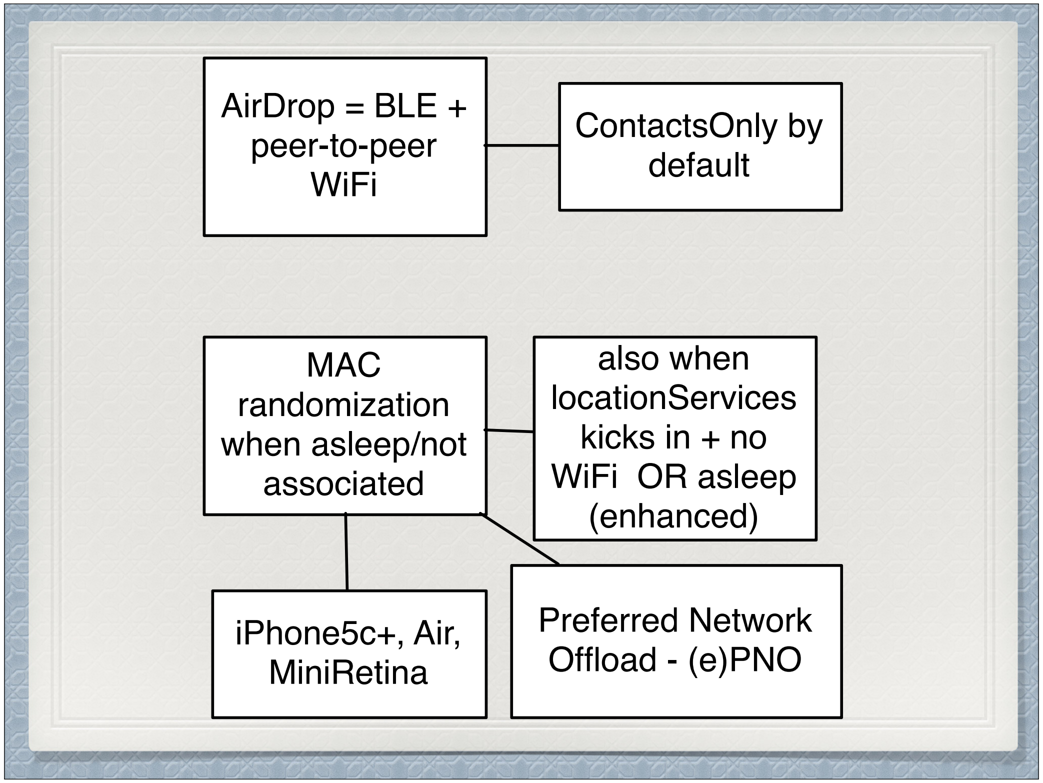
MAC randomization when asleep/not associated

also when locationServices kicks in + no WiFi OR asleep (enhanced)

iPhone5c+, Air, MiniRetina

Preferred Network Offload - (e)PNO

Other networking-related notes - Up top we have AirDrop, which is similar to iMessage in that you have apple as a clearing-house or middleman identity broker in the way that you're shown the contact picture from your address book if the appleID associated with the device is in your contacts, and that info is discovered over bluetooth low energy, and the default being contacts only for that feature is the purest example of convenience vs. safety/privacy, since easy exfiltration of data and metadata being advertised over multicast is probably not something you want on by default if you’re very concerned about security, but hey, contacts only. On the Mac I'm pretty sure that's set to no one, and I know companies that disable the framework itself on Macs by default



AirDrop = BLE + peer-to-peer WiFi

ContactsOnly by default

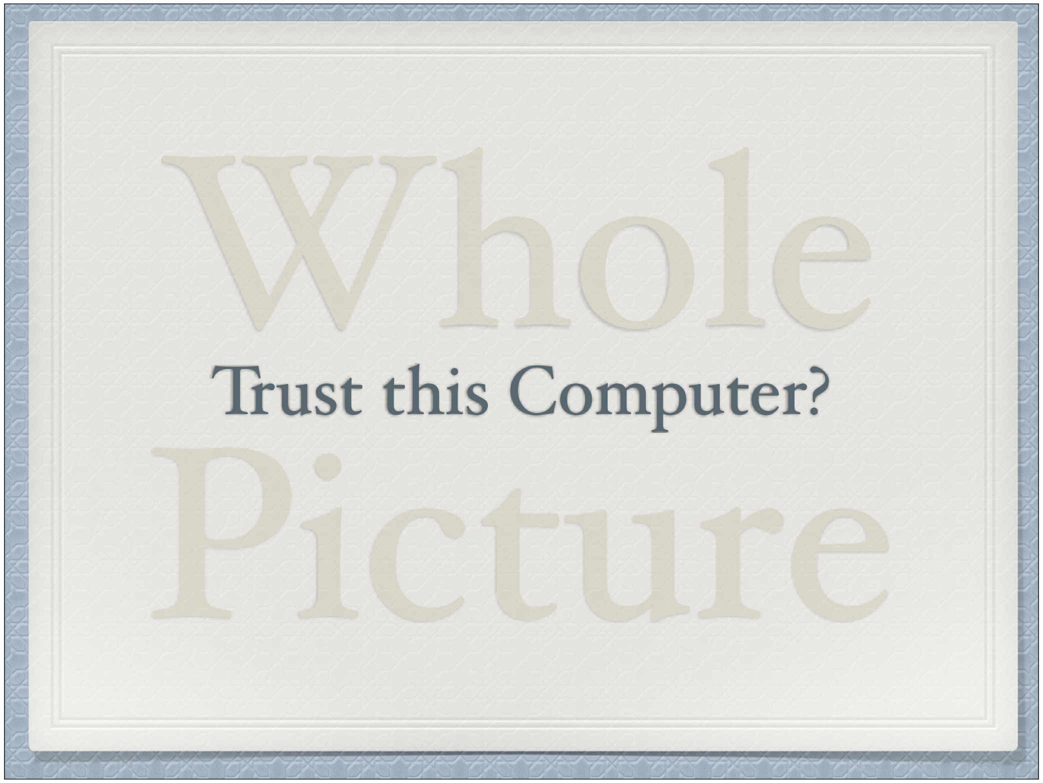
MAC randomization when asleep/not associated

also when locationServices kicks in + no WiFi OR asleep (enhanced)

iPhone5c+, Air, MiniRetina

Preferred Network Offload - (e)PNO

- And finally a counter-measure to being able to track devices that aren't yet associated with a network called MAC address randomization kicks in during very specific circumstances on specific devices, which I was interested in because I wanted to know how far along a handshake a device was before showing its true MAC, as there was an issue years ago with bad DHCP behavior depleting pools on high-volume networks.



# Whole Picture

## Trust this Computer?

<blackout>The biggest takeaway from our book should be to be absolutely sure that the cable you connect to an iOS device is actually where you think it is, tapping Trust when you're not sure can be an invite to be exploited, even for non-jailbroken devices.

# All is Illuminated...

*(Q&A?)   (Apologies to Magritte and Safran Foer)*

If you say I've imparted all the wisdom you need over the course of this preso, I'll call you a liar, but I'll open it up for Questions, and anything we don't have time for I'll just point to the amazon link

# Thanks!

http://url.aru-b.com/packtbook

in *Allister Banks* *sacrilicious*

*al@aru-b.com*