CORELLIUM

The hidden gems

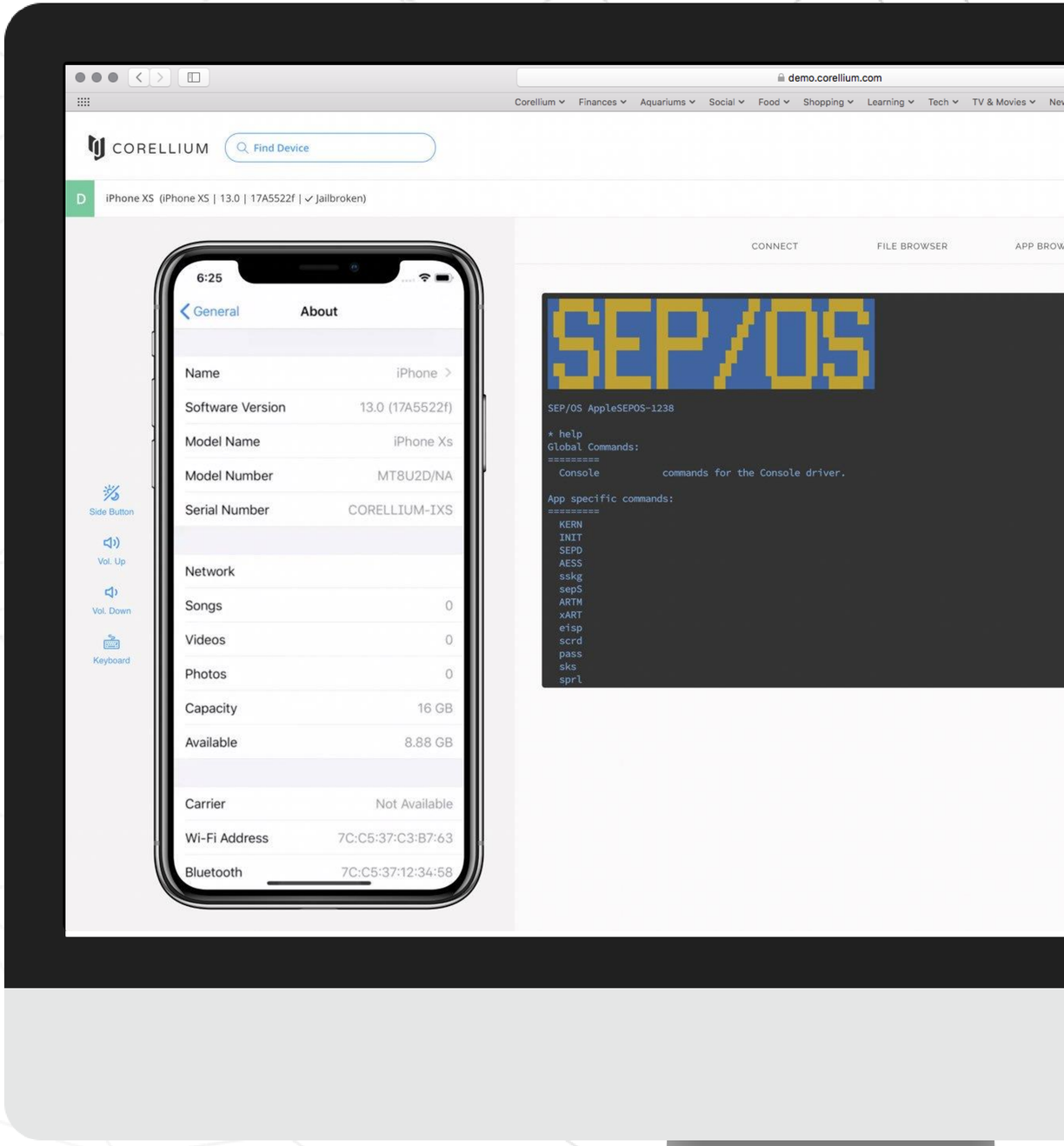
Blackhat 2019

About Corellium

Our CORSEC product is designed for research, automated testing and fuzzing.

Fidelity is our main goal.

We run real iOS – with real bugs that have real exploits.

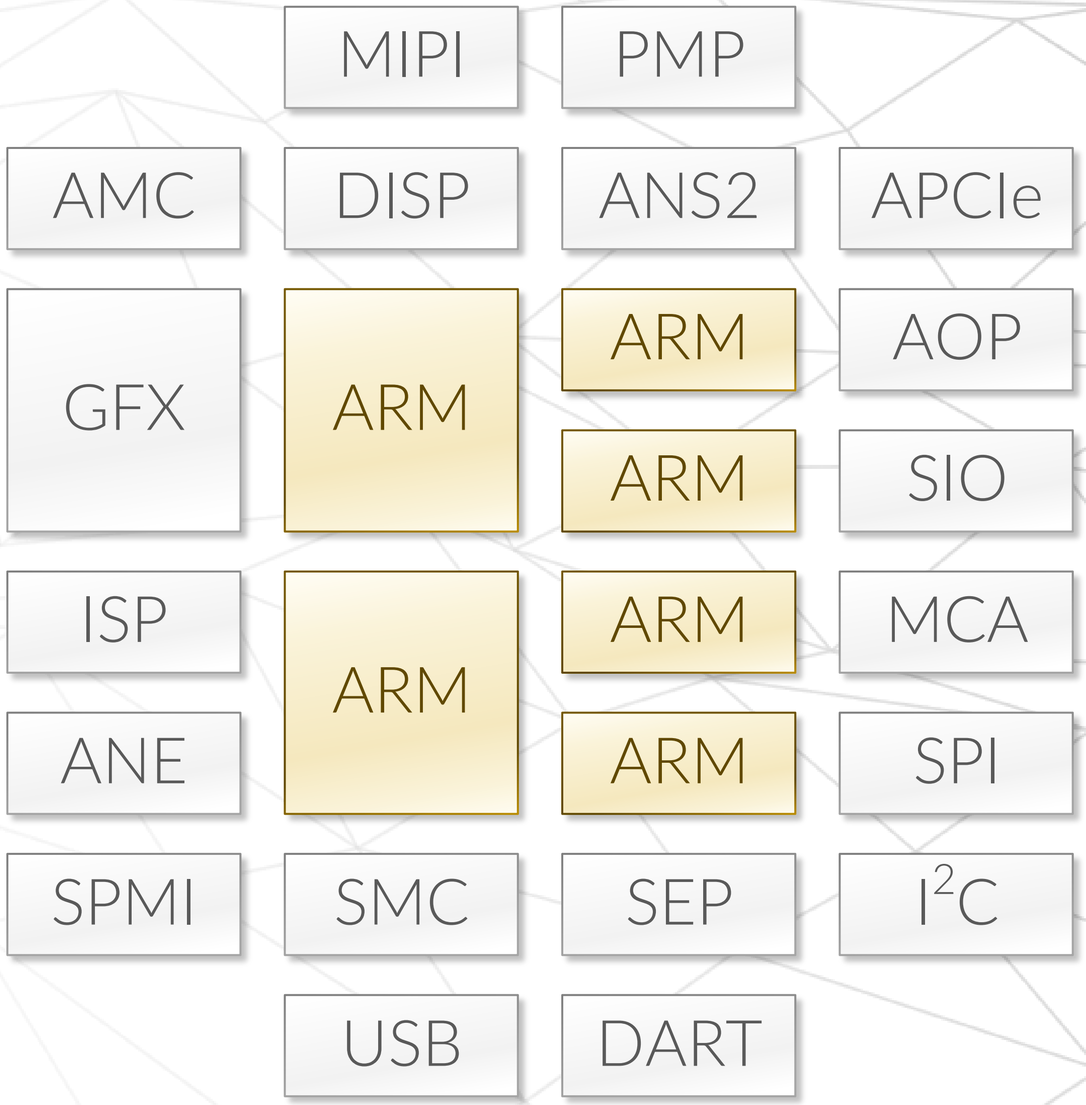


Our models

The processor is virtualized; peripherals are generally emulated at register level.

Apple products have unique hardware; the ARM cores are cutting-edge and the peripherals are designed in-house.

The research that makes our models possible is the cornerstone of our product.



The hidden gems of iOS

XS/XR: star of the show

- Most security improvements use new A12 hardware
- Driven by availability of PAC and improved SEP
- Features unique to Apple products

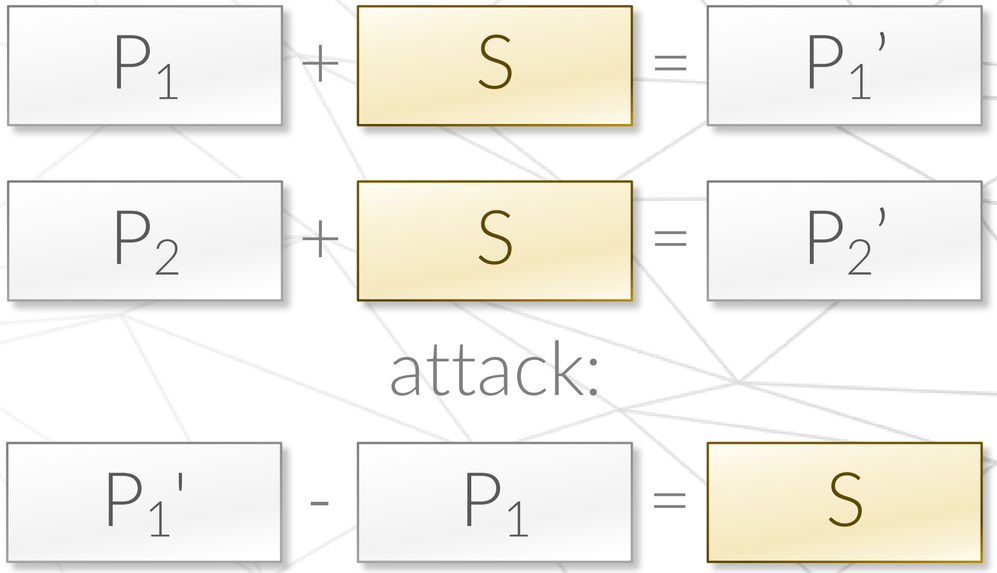
Old chips & new features

- New user-facing features available across the board
- Support goes back to 2014: A8 in the iPad 4

Intro to PAC

(Pointer Authentication Codes)

- **Pointer compromises**
  - If you could write something in the kernel memory…
  - … you'd probably write a pointer!
  - ROP, object substitution, classic stack attacks
- **Need to predict pointers to attack**
- **ASLR: a first attempt at defense**
  - Need secret information to predict pointers
  - Fortunately, it's easy to get the secret if you know just one pointer!



Intro to PAC (2)

- **Cryptographically secure the secret**
  - Hardware implementation makes it fast
  - Compiler support makes it pervasive
  - Special-purpose pointer wrapping opcodes
  - Reuse data/instruction abort mechanism for failures
- **Universal fast crypto primitive**
  - Non-readable keys on Apple hardware
  - PACGA opcode for non-pointer applications

Inverse of F can't be calculated

The iOS 12 JOP hash

- **First use of PACGA in iOS**
- **Control flow integrity verification**
  - Calculated and saved on exception entry
  - Recalculated and compared on exception return
  - Hash of SP_EL0, SPSR_EL1, ELR_EL1 and LR (EL0)
- **Very low overhead**
  - Defeats attacks on exception return state

```
jop_hash_save:
    pacga       x1, x1, x0                  ; CODE XREF=sub_fffffff007b1dee8
    and         x2, x2, #0xffffffffdfffffff
    pacga       x1, x2, x1
    pacga       x1, x3, x1
    str         x1, [x0, #0x128]
    ret

--------------------------------------------------------------------------------

jop_hash_check:
    pacga       x1, x1, x0                  ; CODE XREF=sub_fffffff007b1dee8
    and         x2, x2, #0xffffffffdfffffff
    pacga       x1, x2, x1
    pacga       x1, x3, x1
    ldr         x2, [x0, #0x128]
    cmp         x1, x2
    b.ne        loc_fffffff0080f9188
--------------------------------------------------------------------------------
    ret

--------------------------------------------------------------------------------

loc_fffffff0080f9188:
    mov         x1, x0                      ; CODE XREF=jop_hash_check+24
    adr         x0, #0xfffffff0080f9194     ; argument #1 for method panic,
    bl          panic                       ; panic

aJopHashMismatc:
    db          "JOP Hash Mismatch Detected (PC, CPSR, or LR corruption)", 0
```

Demo

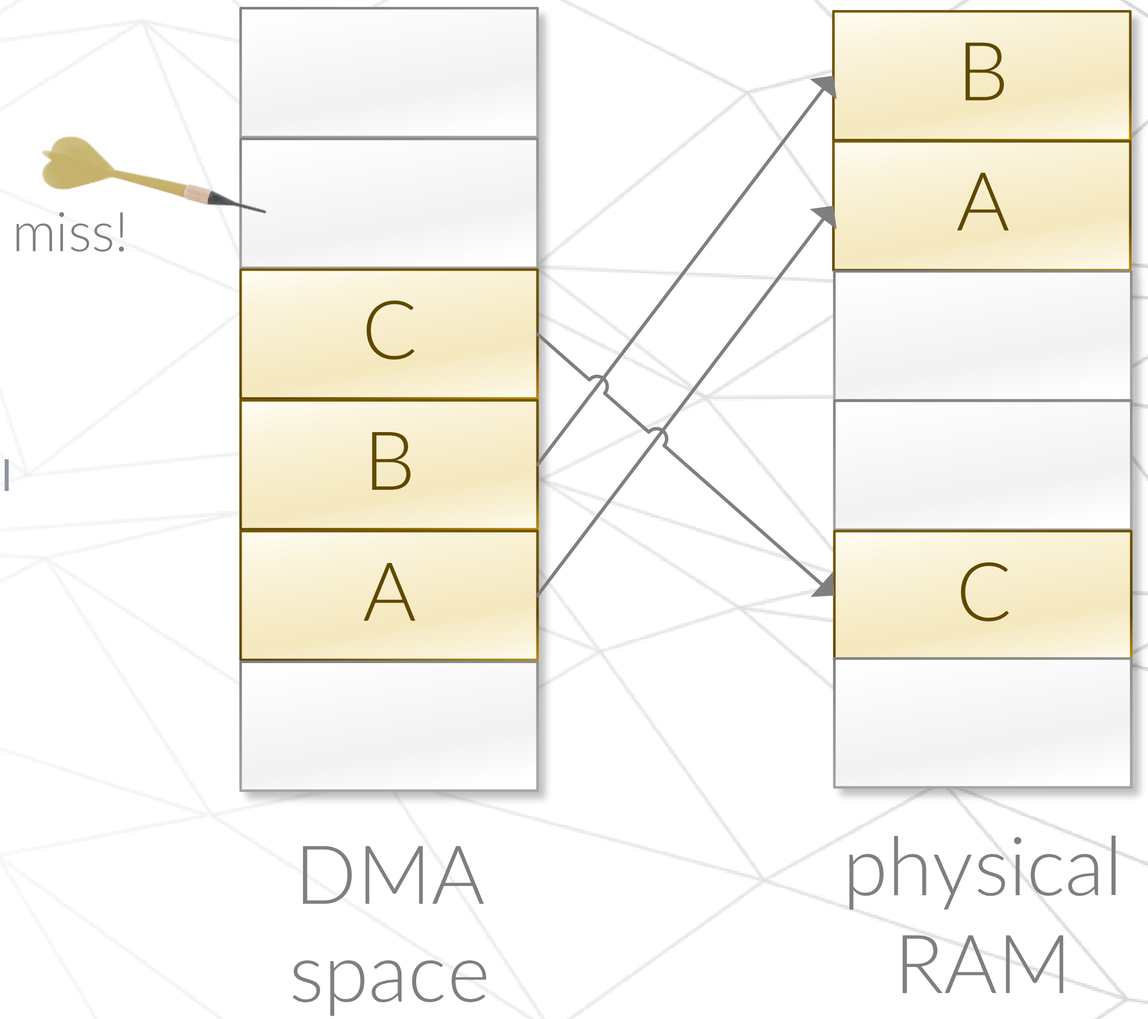
JOP hash bypass

Find thread with CoreTrace
Find CPU state and JOP hash in thread
Modify thread CPSR or PC
Take a live snapshot
Resume and observe panic
Restore the snapshot
Patch out JOP hash checks

DARTs

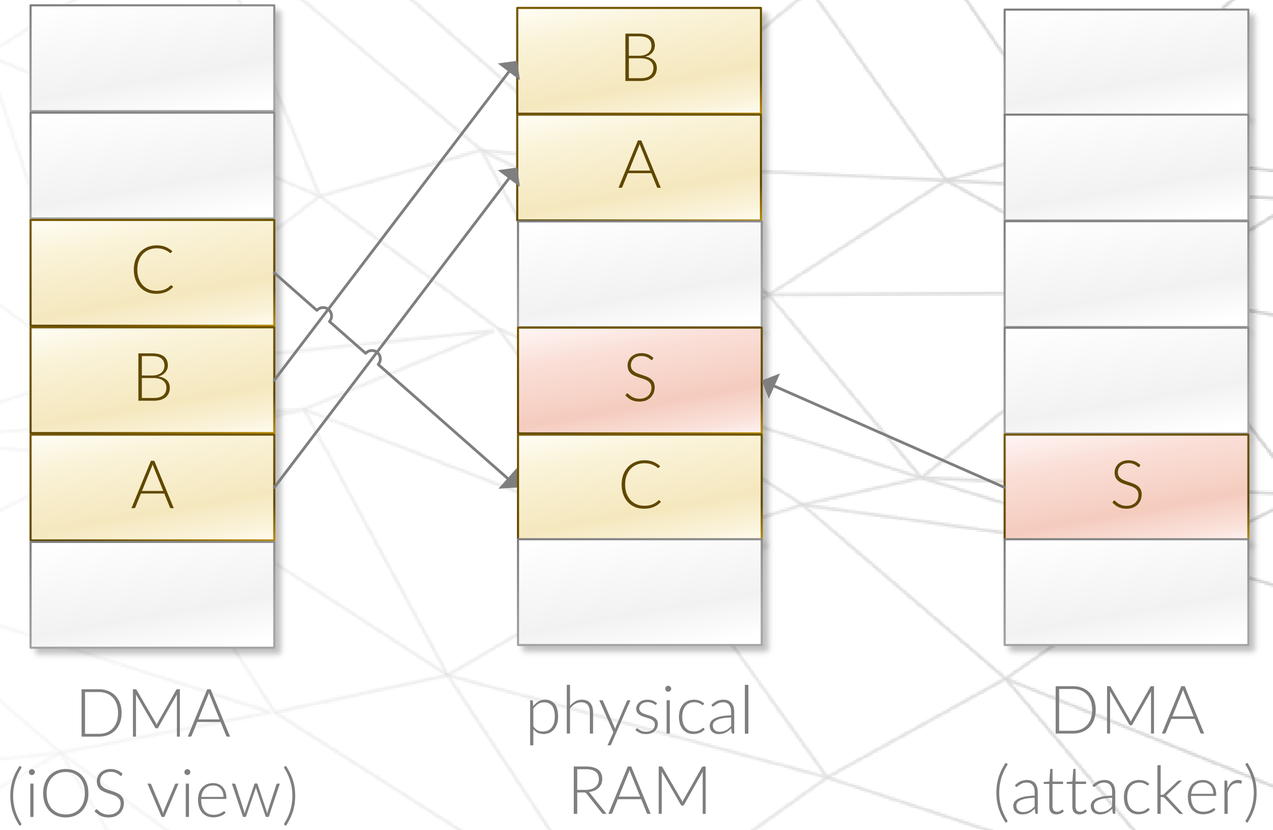
(DMA Address Remap Table)

- **Apple’s take on IOMMUs**
  - Concept came from GPUs: GARTs for texture gather
  - In fact, PowerVR GPUs and classic video decoders still have their own GARTs
- **Security by address space isolation**
- **Multiple address spaces for one I/O device**
  - Most devices use only one space
  - But display isolates command and data streams



DARTs (2)

- **Significant security impact**
  - A DART protects RAM from PCIe access
  - A DART limits USB attack surface (on A12)
  - Badly set up DARTs have exposed SEP on A10
- **DARTs configured by AP via MMIO**
- **Escalation from kernel write**
  - Write MMIO to reconfigure DART
  - Use PCIe (NVMe, baseband, WLAN/BT) or IOP exploit to take control



DART protection in iOS 13

- **On A12, kernel maps MMIO via PPL**
  - Configured by new entries in device tree
  - MMIO for IOMMUs (DART, SART, PCIe) treated in a special way
- **Kernel verifies DART MMIO map on first use**
  - DART driver checks that PPL locked down the MMIO range
  - Panic if entry not found or not matching

Demo

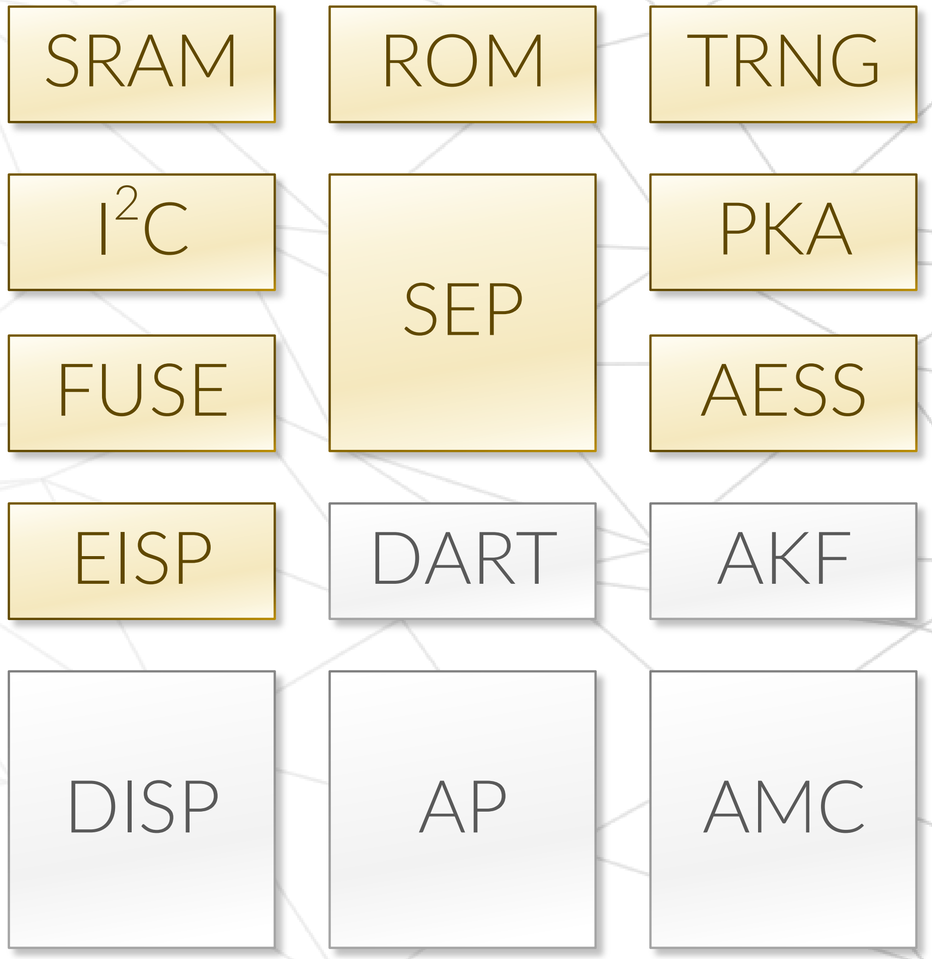
DART protection

Use kernel debugger to break on first access
Investigate memory map
Show table of IO map entries

SEP hardware

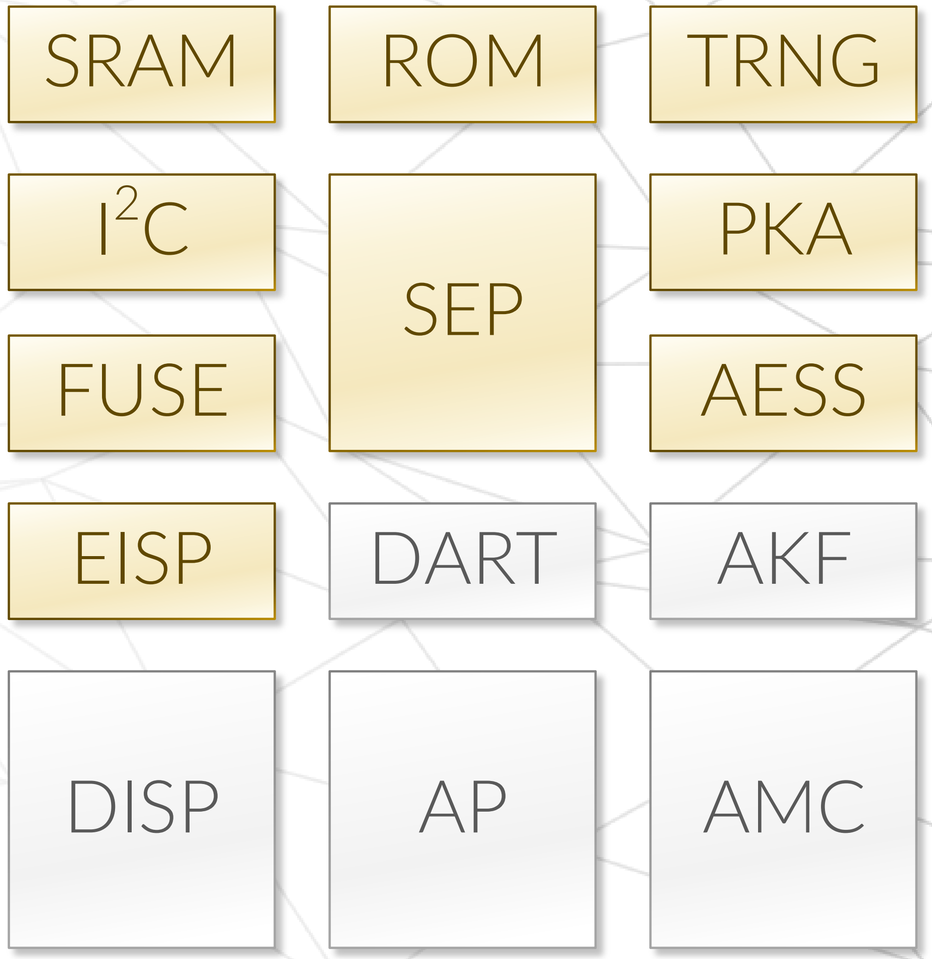
(Secure Enclave Processor)

- **Separate CPU core**
  - 32-bit Cortex-A7 on A8-A10
  - 64-bit Apple CPU on A11
  - 64-bit Apple CPU with PAC on A12, behind a DART
- **Separate peripherals**
  - Common address space, though
- **Entirely security focused**
  - Lots of other CPU cores to do other things
  - No need to share CPU with them



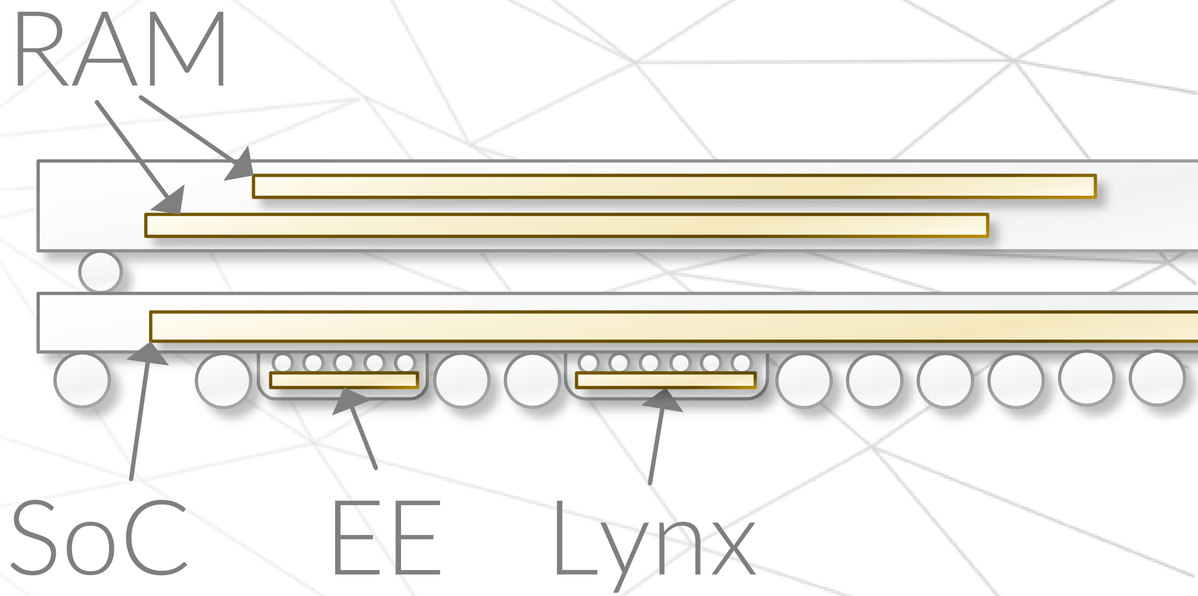
SEP peripherals

- **Secret key vault in AESS**
  - Encryption, decryption and CMAC
  - Fused keys not extractable even by SEP
- **Another secret in PKA**
  - Used for iCloud authentication
  - Also not extractable by SEP
- **True RNG**
  - Entropy source + AES NIST CTR-DRBG
  - Verified operational on boot
- **Secure fuse box**



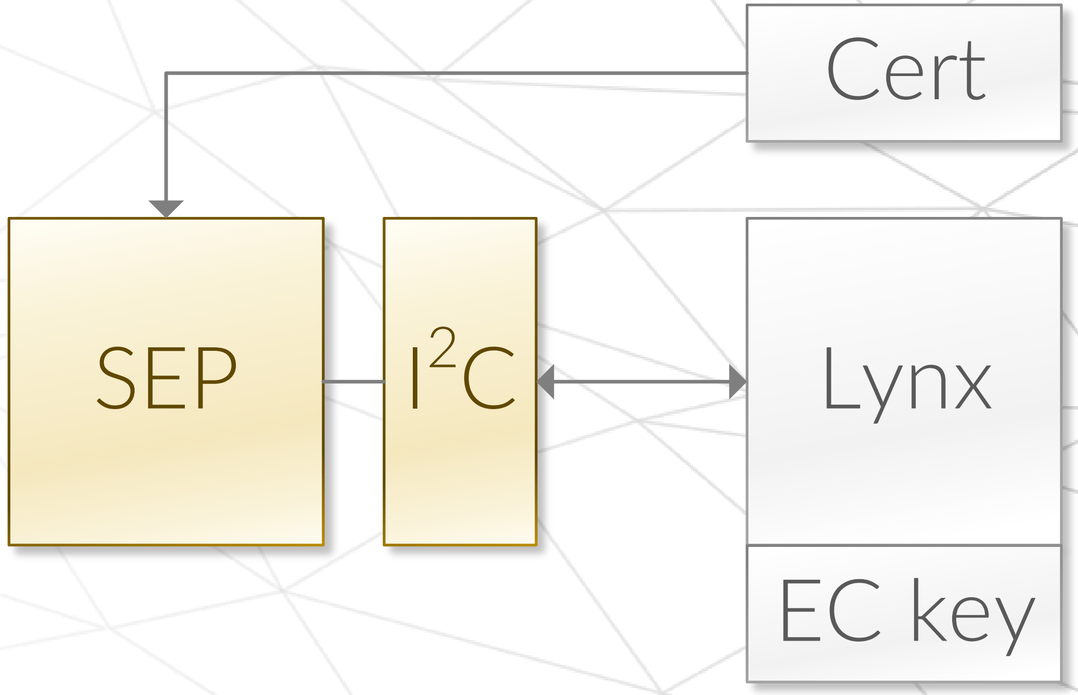
SEP peripherals (2)

- **Camera interface on A11/A12**
  - With its own Image Sensor Processor for Face ID
- **I²C bus**
  - EEPROM used for anti-rollback protection
  - Integrated on package since A11
  - TSMC **InFO** technology makes probing impractical
- **xART on iOS 13 / mART on older**
  - This is the information protected by the EEPROM



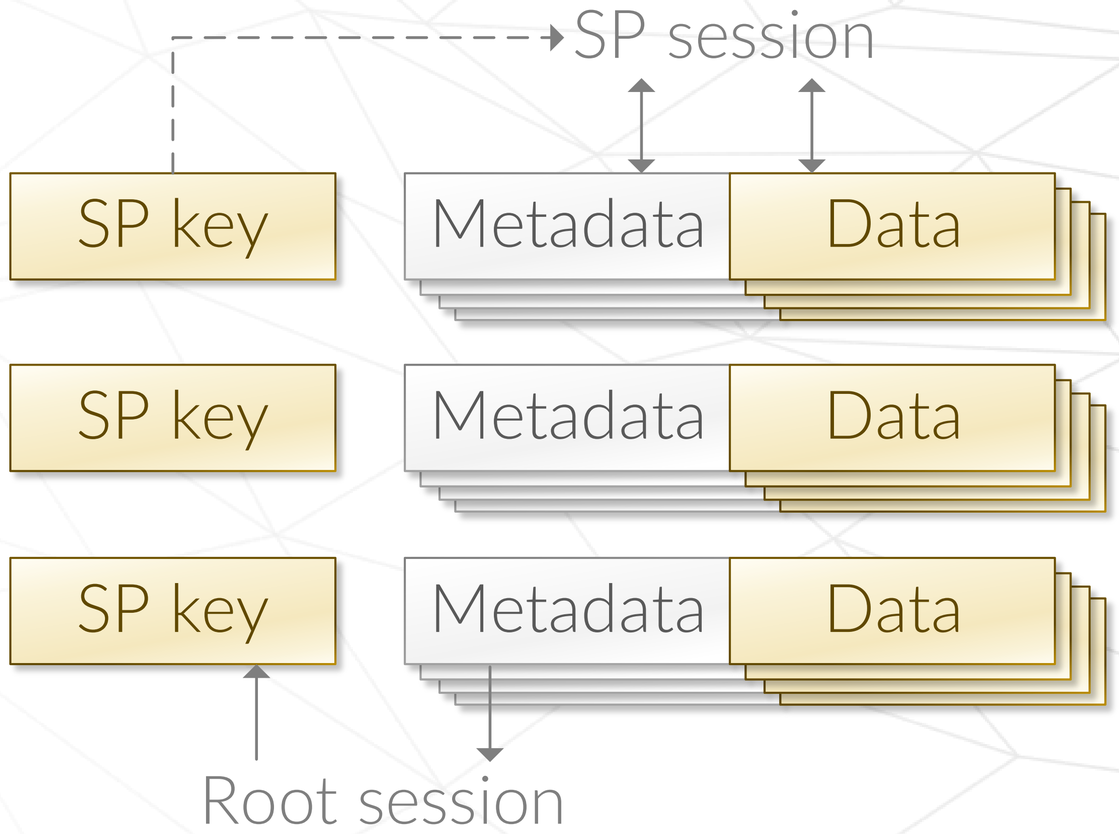
The secretive Lynx

- **Custom chip on I²C inside A12 CPU**
  - Securely stores anti-rollback counters
- **Derivative of STSAFE-A100**
  - Unfortunately, the Apple firmware has completely different command set and structure.
- **Lynx and SEP pair during restore**
  - SEP trusts Lynx based on public key certificate.
  - Lynx makes information available only with preshared key.



The secretive Lynx (2)

- **ECDH based sessions over I²C**
  - Root session can only query metadata, set SP keys and create SP sessions based on these keys
- **Each SP session has access to one SP locker**
  - Changing SP key causes locker content to be lost.
  - Each SP locker contains metadata and data.
  - Stored in four copies to guarantee atomic write: one copy damaged during write leaves three copies to perform a majority vote.
  - Metadata is readable, but not writable, without SP key.



Demo

SEP & Lynx

Use SEP debugger to connect to SEP
Query Lynx contents using Corellium debug stub

SEP in iOS 13

- **New LLVM version used**
- **Function outlining enabled**
  - Extracts shared pieces of assembly into “functions”.
  - Even function prologues or epilogues are not safe.
  - Confuses both disassemblers and decompilers.
- **Debugger helps understand code paths**
- **No console included on new SEPOS**

```
sub_1bf84:
    pacibsp
    stp         x24, x23, [sp, #-0x40]!
    stp         x22, x21, [sp, #0x10]
    stp         x20, x19, [sp, #0x20]
    stp         x29, x30, [sp, #0x30]
    add         x29, sp, #0x30
    sub         sp, sp, #0x10
    mov         x22, x3
    ...
    sub         sp, x29, #0x30
    ldp         x29, x30, [sp, #0x30]
    b           sub_1c30c
```

Function (abbreviated)

```
sub_1c30c:
    ldp         x20, x19, [sp, #0x20]
    ldp         x22, x21, [sp, #0x10]
    ldp         x24, x23, [sp], #0x40
    retab
```

Outlined epilogue (shared with 2nd function)

SEP in iOS 13 (2)

- **Secure Enclave API now used by OS itself**
  - Allows creating non-exportable (except wrapped) keys in SEP on behalf of ordinary EL0 software.
- **Around in iOS 9, used by 3rd parties**
  - We model the Secure Enclave API at SEP level.
  - In iOS 13 we've seen requests to it during first boot of iOS, marking created keys as "system".
  - Extensible API based on DER blobs: possibly new key types in the future?

```
2000000000000011                111 SET
  2000000000000010               24 SEQUENCE
    000000000000000c              4 UTF8:"acmh"
    0000000000000004             16 OCT:37 4D 82 E5...
  2000000000000010                7 SEQUENCE
    000000000000000c              2 UTF8:"bc"
    0000000000000002              1 INT:8
  2000000000000010               56 SEQUENCE
    000000000000000c              2 UTF8:"ed"
    2000000000000011             50 SET
      2000000000000010           48 SEQUENCE
        000000000000000c          3 UTF8:"acl"
        2000000000000011         41 SET
          2000000000000010        8 SEQUENCE
            000000000000000c      3 UTF8:"ock"
            0000000000000001      1 BOOL:TRUE
          2000000000000010        9 SEQUENCE
            000000000000000c      4 UTF8:"odel"
            0000000000000001      1 BOOL:TRUE
          2000000000000010        9 SEQUENCE
            000000000000000c      4 UTF8:"osgn"
            0000000000000001      1 BOOL:TRUE
          2000000000000010        7 SEQUENCE
            000000000000000c      2 UTF8:"oa"
            0000000000000001      1 BOOL:TRUE
  2000000000000010                7 SEQUENCE
    000000000000000c              2 UTF8:"kt"
    0000000000000002              1 INT:5
  2000000000000010                7 SEQUENCE
    000000000000000c              1 UTF8:"o"
    000000000000000c              2 UTF8:"oc"
```

iBoot in iOS 13

- **Minimal changes**
- **Now preloads SEP and other IOPs**
  - iOS used to load them in previous versions.
  - Now, only external peripherals like MultiTouch are loaded in iOS.
  - iBoot loads all IOP firmware, checks validity of IMG4 and parses the load segments of the Mach-O file inside.
- **More hardware lock-downs for IOPs**

iBoot in iOS 13 (2)

- **Each IOP has a MMIO range**
  - Mostly used to configure bus access, performance and sleep settings, etc.
  - Offers access to (some?) System Registers of the IOP's ARM core!
  - This includes **RVBAR_EL3** which controls entry point after reset.
- **RVBAR_EL3 is now locked down**
  - iBoot loads the firmware, so it knows where RVBAR_EL3 should be set to.
  - Can't subvert reset vectors on IOPs anymore.

THANK YOU

Corellium 2019