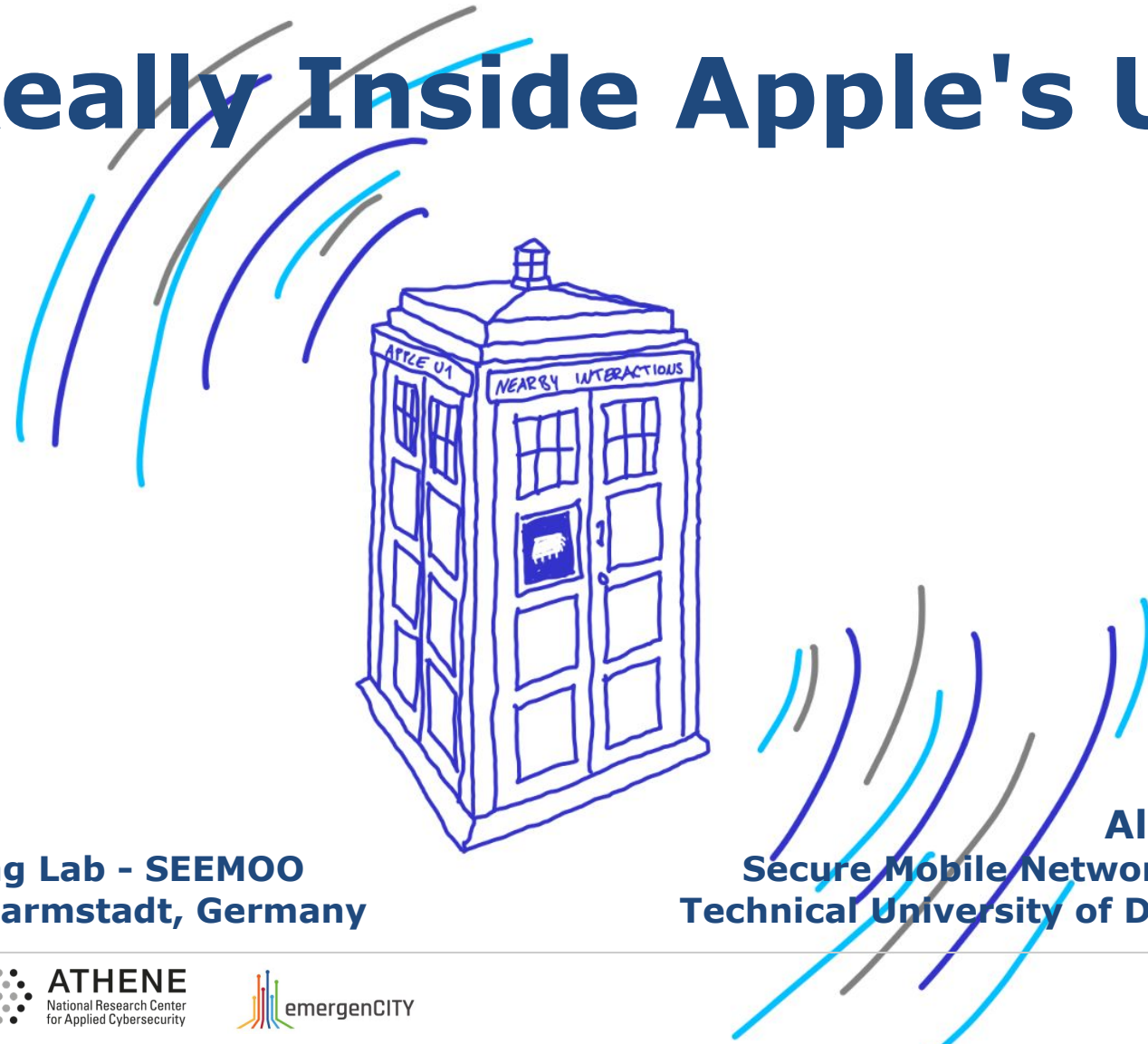


Wibbly Wobbly, Timey Wimey

What's Really Inside Apple's U1 Chip



Jiska Classen
Secure Mobile Networking Lab - SEEMOO
Technical University of Darmstadt, Germany

Alexander Heinrich
Secure Mobile Networking Lab - SEEMOO
Technical University of Darmstadt, Germany

Ultra Wideband (UWB) U1 Chip

Nobody knows what
it is or does

Non-interceptable with
cheap SDRs



Must be
hacker-proof!

Only available in the latest
generation of devices

[Table of Contents](#) 

Ultra Wideband security in iOS

The new Apple-designed U1 chip uses Ultra Wideband technology for spatial awareness — allowing iPhone 11, iPhone 11 Pro and iPhone 11 Pro Max or later iPhone models to precisely locate other U1-equipped Apple devices. Ultra Wideband technology uses the same technology to randomise data found on other supported Apple devices:

- MAC address randomisation
- Wi-Fi frame sequence number randomisation

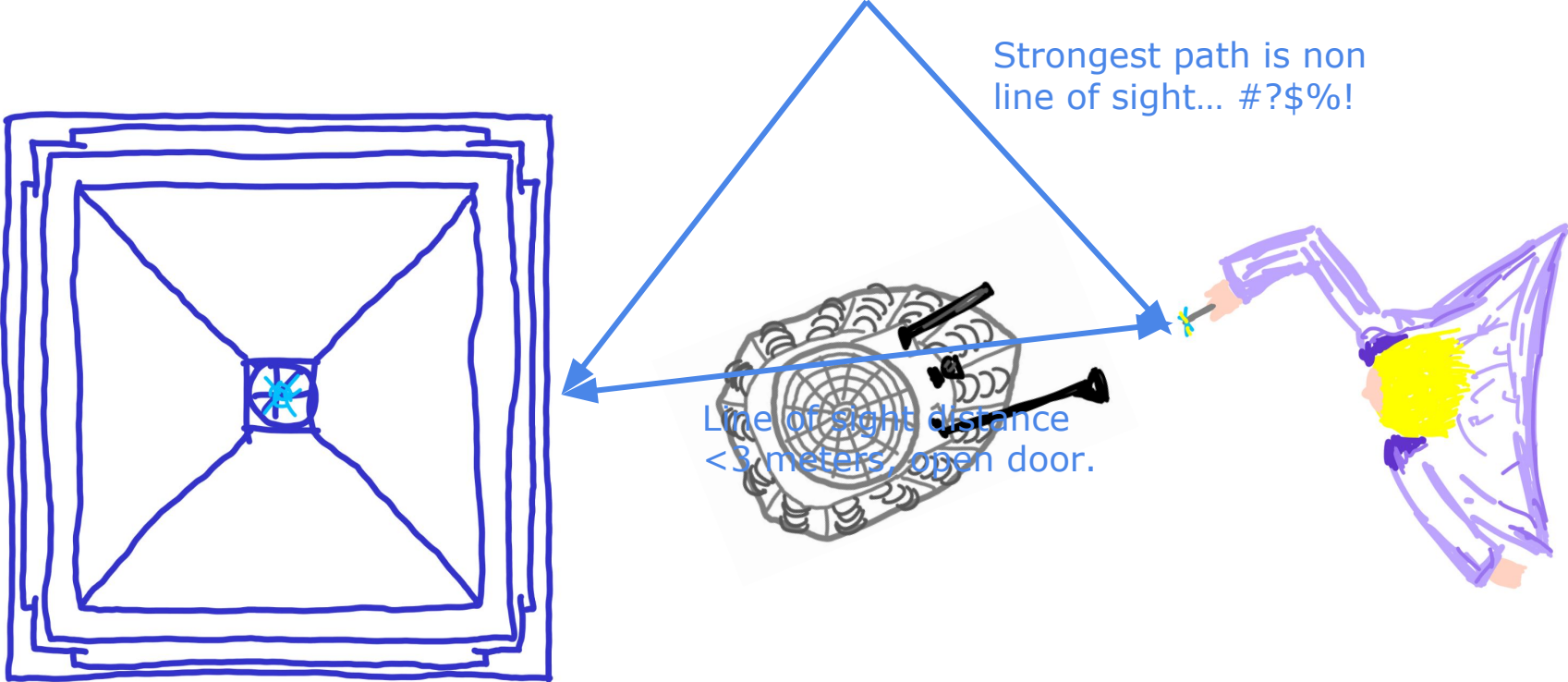


Published Date: 18 February 2021

See also

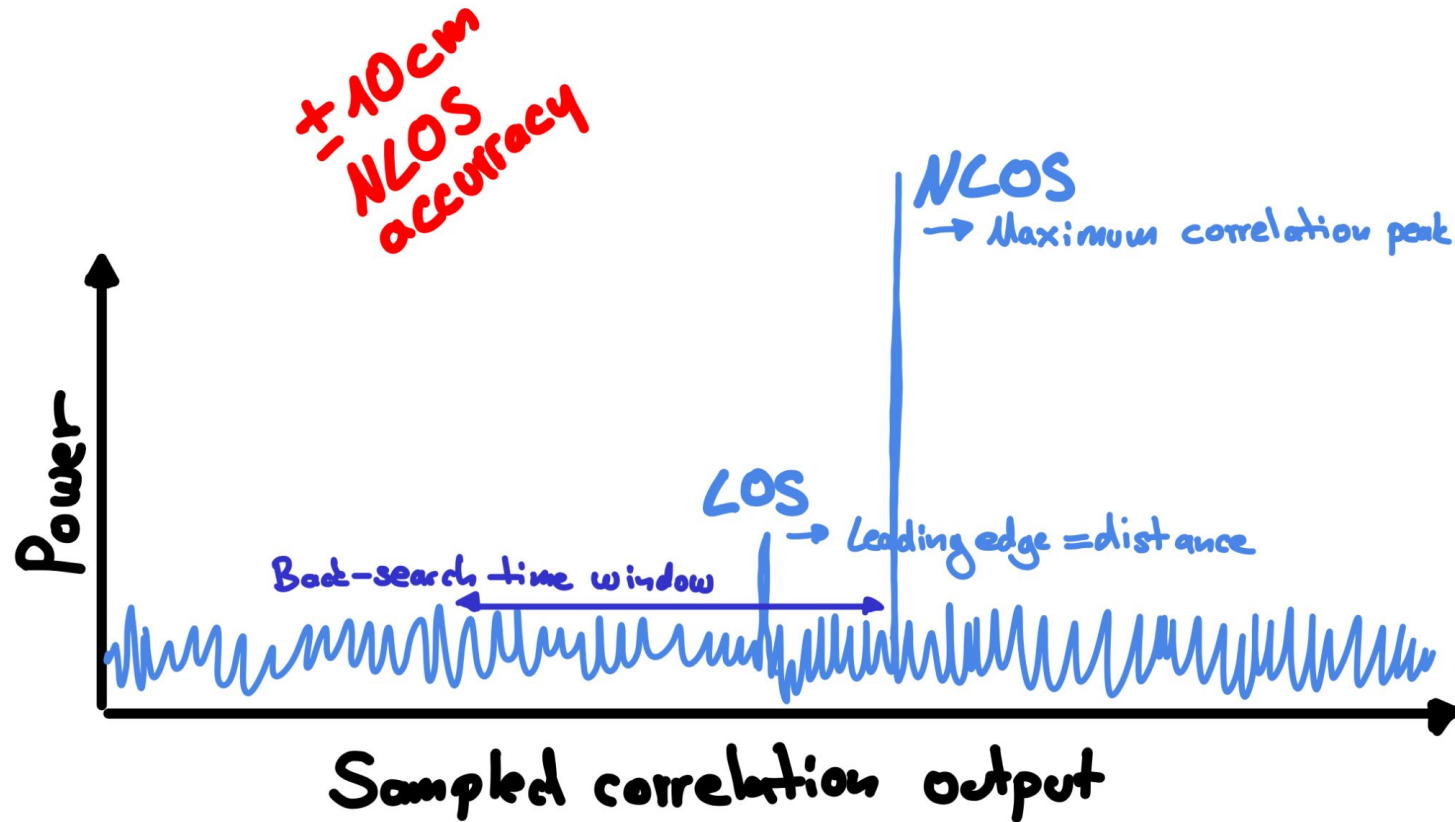
[Wi-Fi privacy](#)

UWB Secure Ranging & NLOS Distance Measurement

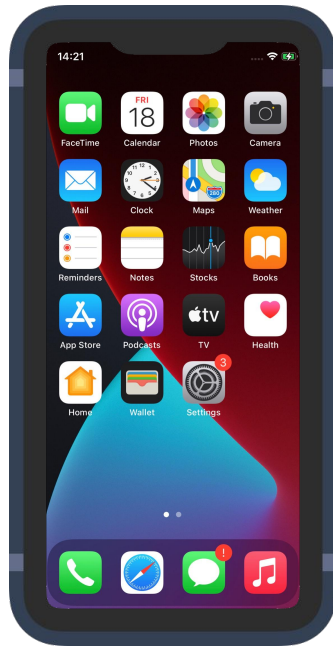


UWB Secure Ranging & NLOS Distance Measurement

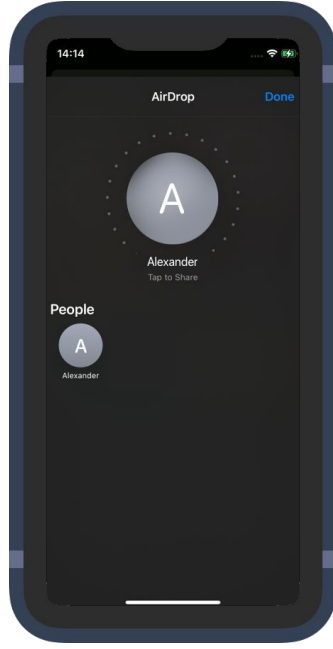
Somewhat



UWB Features

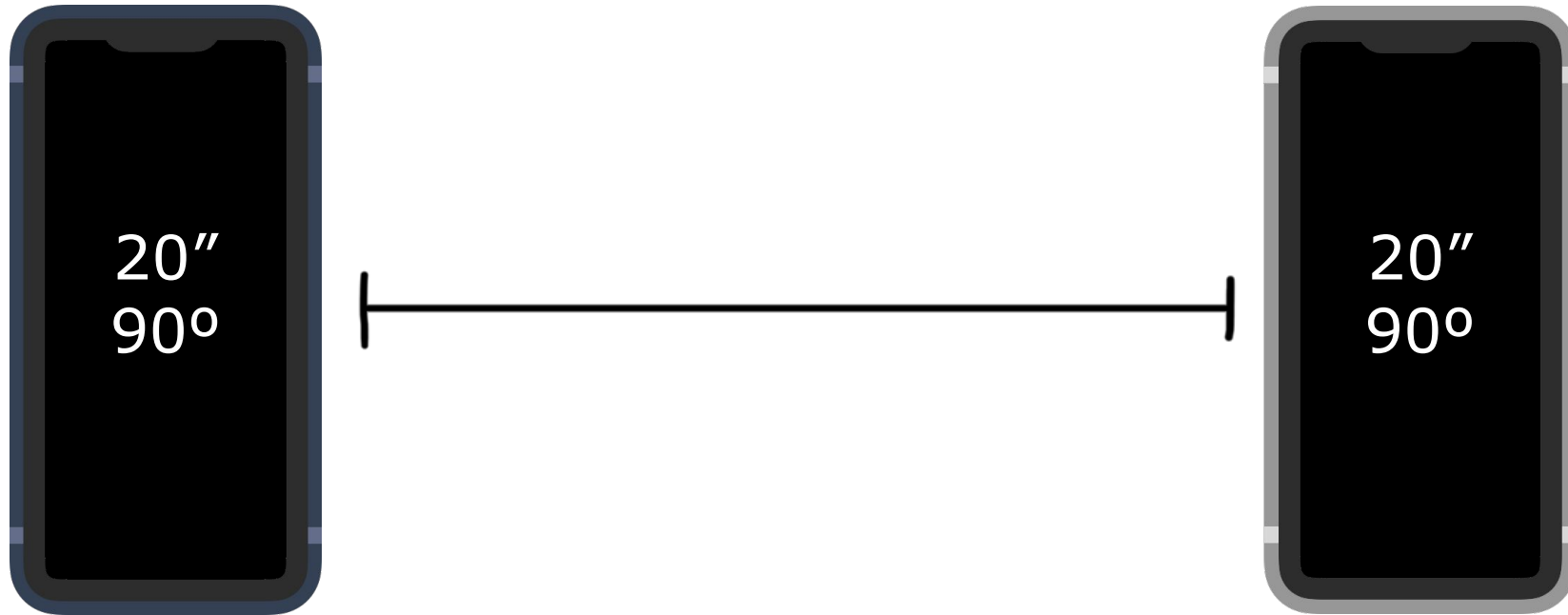


UWB

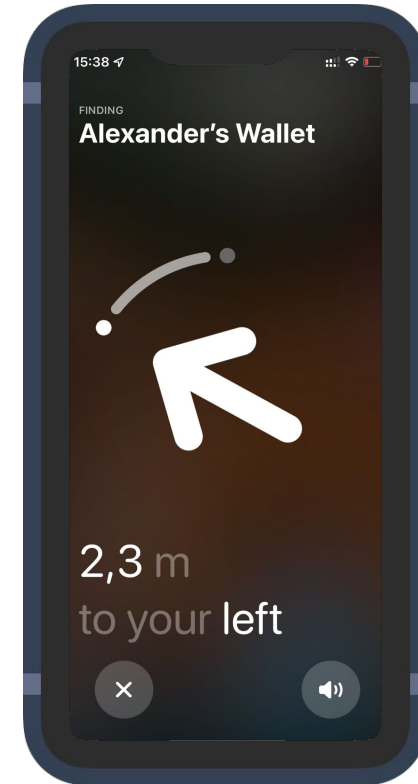


Distance 1"
Angle 0°

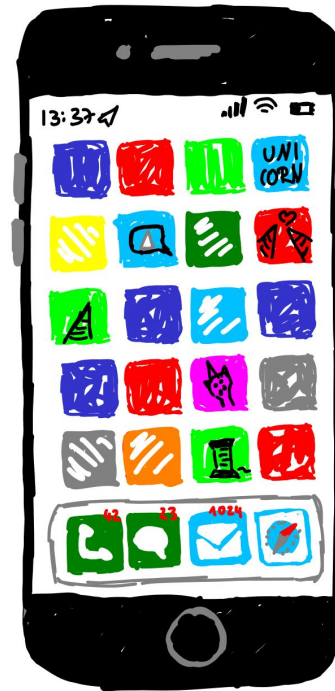
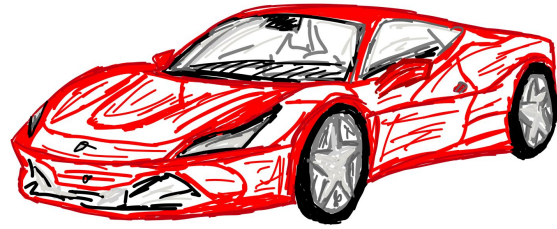
Nearby Interaction



Find My



UWB to X



UWB Internals

AirDrop



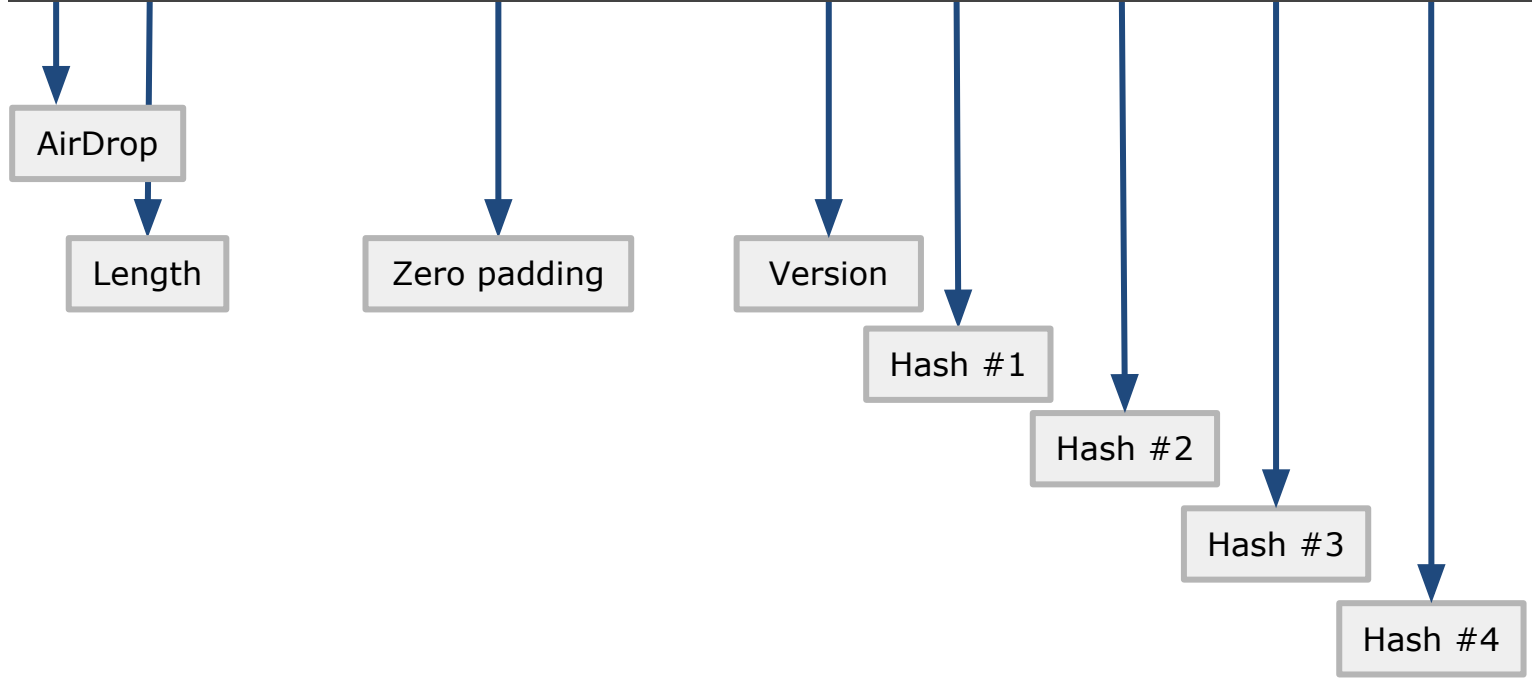
UWB Beacons
+
AirDrop Protocol

AirDrop Bluetooth Discovery



Random **non-resolvable**
MAC address

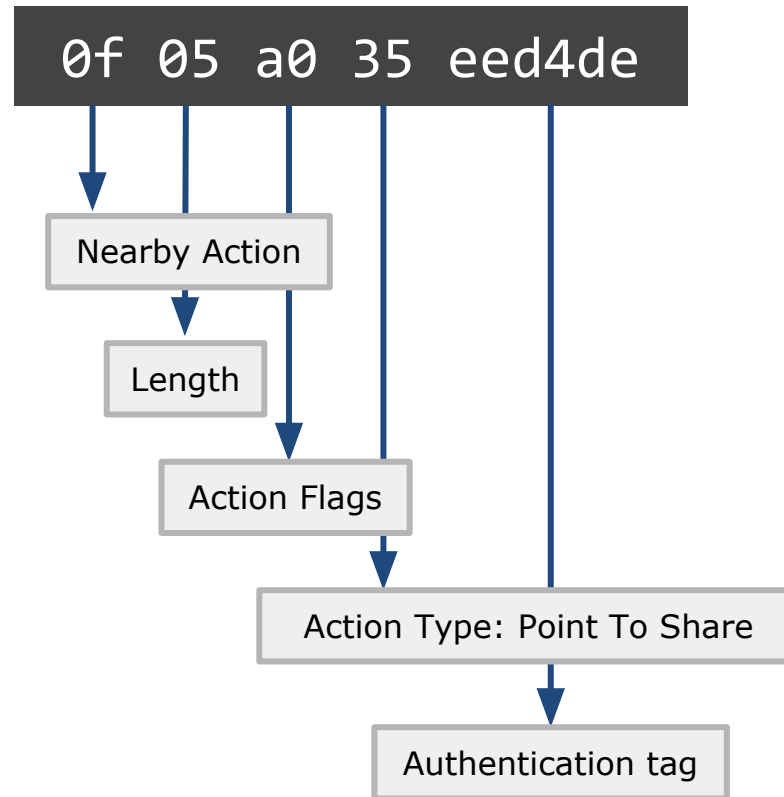
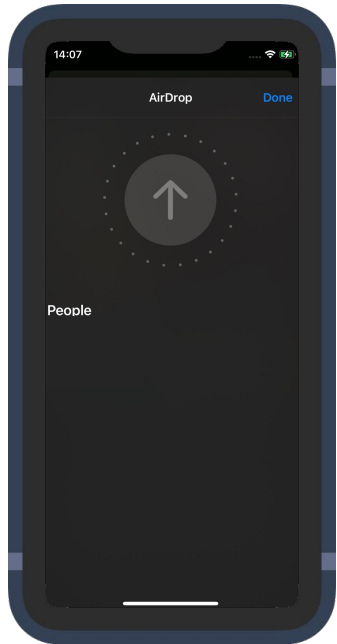
05 12 00000000 00000000 01 ac5b d44e a87b dafe 00




UWB Bluetooth Discovery



Random **resolvable** MAC address



UWB Bluetooth Discovery

 MAC address

sharingd

0f 05 a0 35 eed4de

✓ Nearby device detected

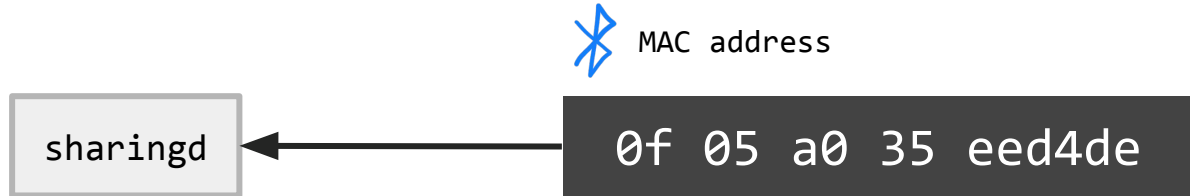
✓ UWB Point To Share



Authentication Tag Validation



- ✓ Nearby device detected
- ✓ UWB Point To Share
- ✓ Validate Auth Tag



$$\text{SipHash}(\text{Bluetooth MAC address}, \text{IRK}) = \text{Auth Tag}$$

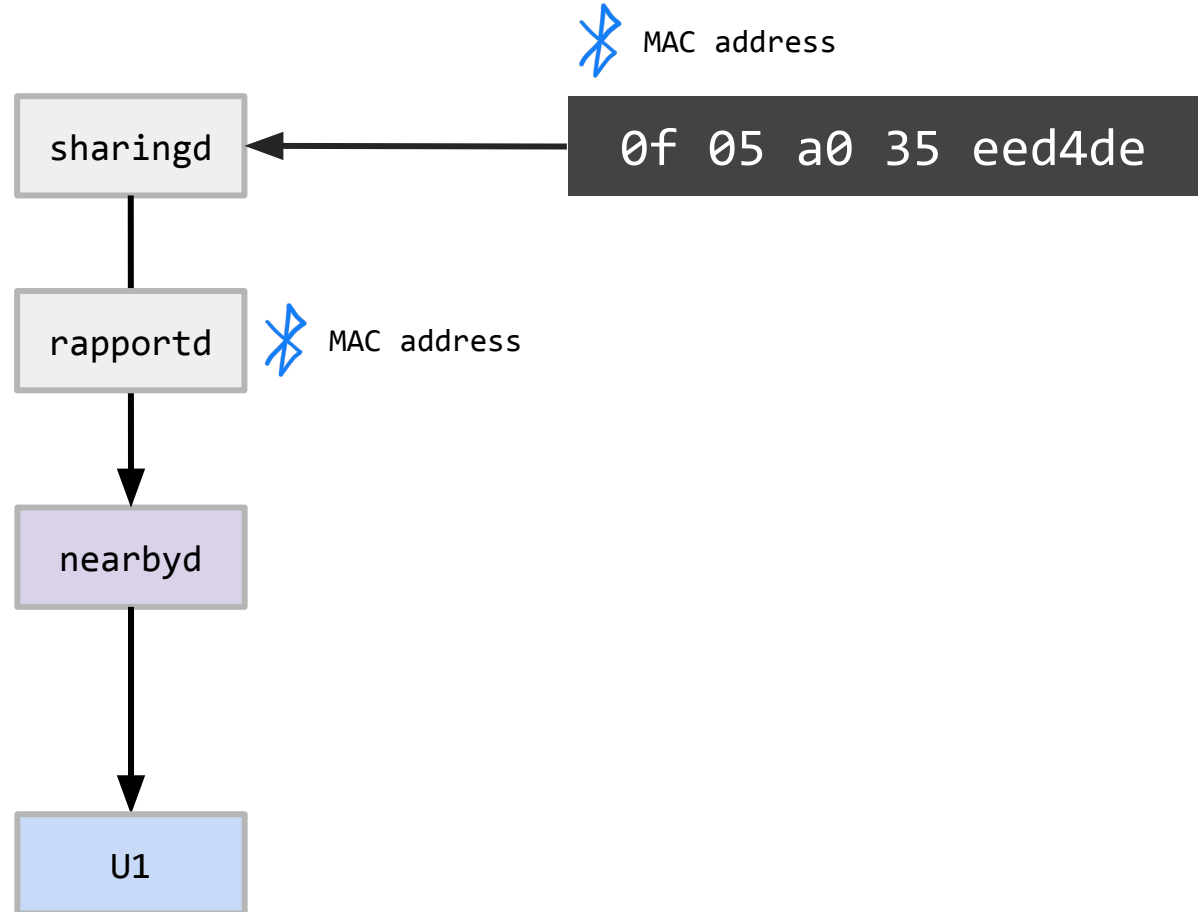
UWB Bluetooth Discovery



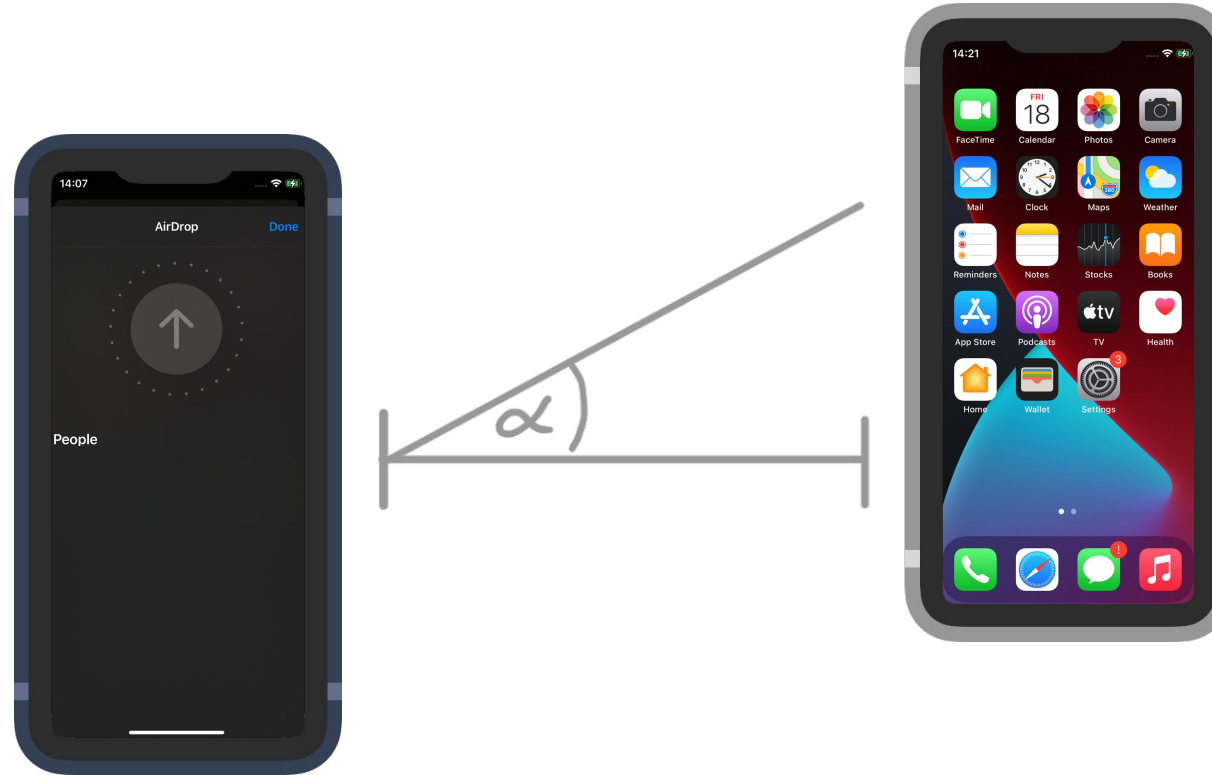
- ✓ Nearby device detected
- ✓ UWB Point To Share
- ✓ Validate Auth Tag

Initiate ranging

Whitelist UWB MAC address



AirDrop Ranging



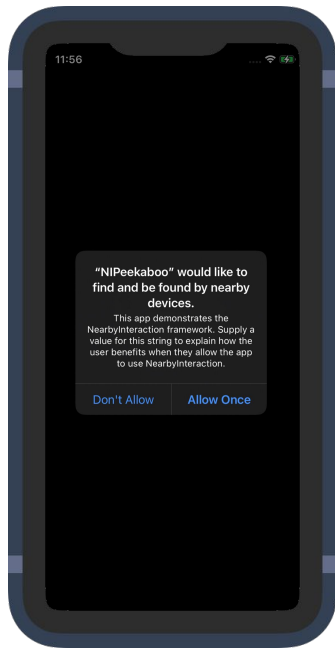
UWB Ranging and Angle measurements

AirDrop Ranging

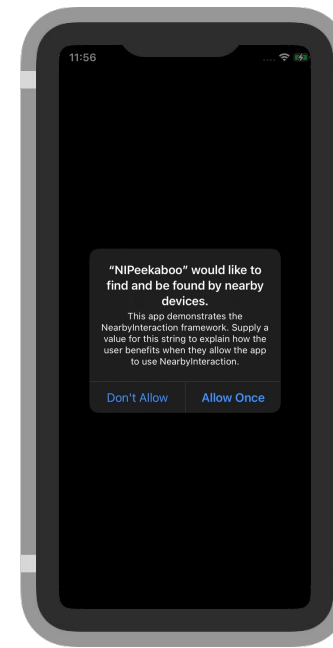


UWB Ranging and Angle measurements

Nearby Interaction Framework

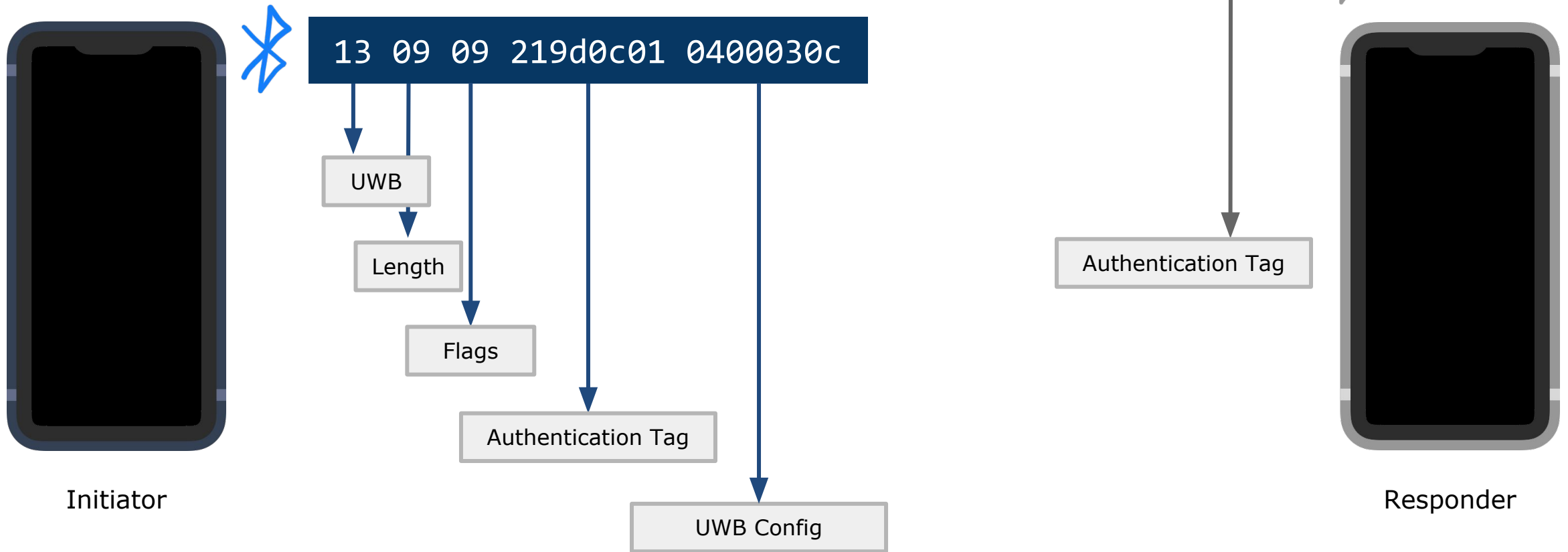


Initiator

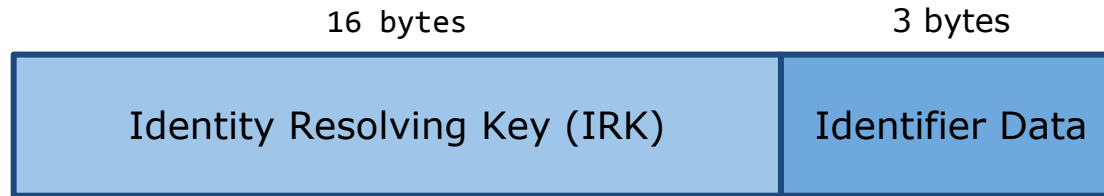


Responder

Bluetooth discovery



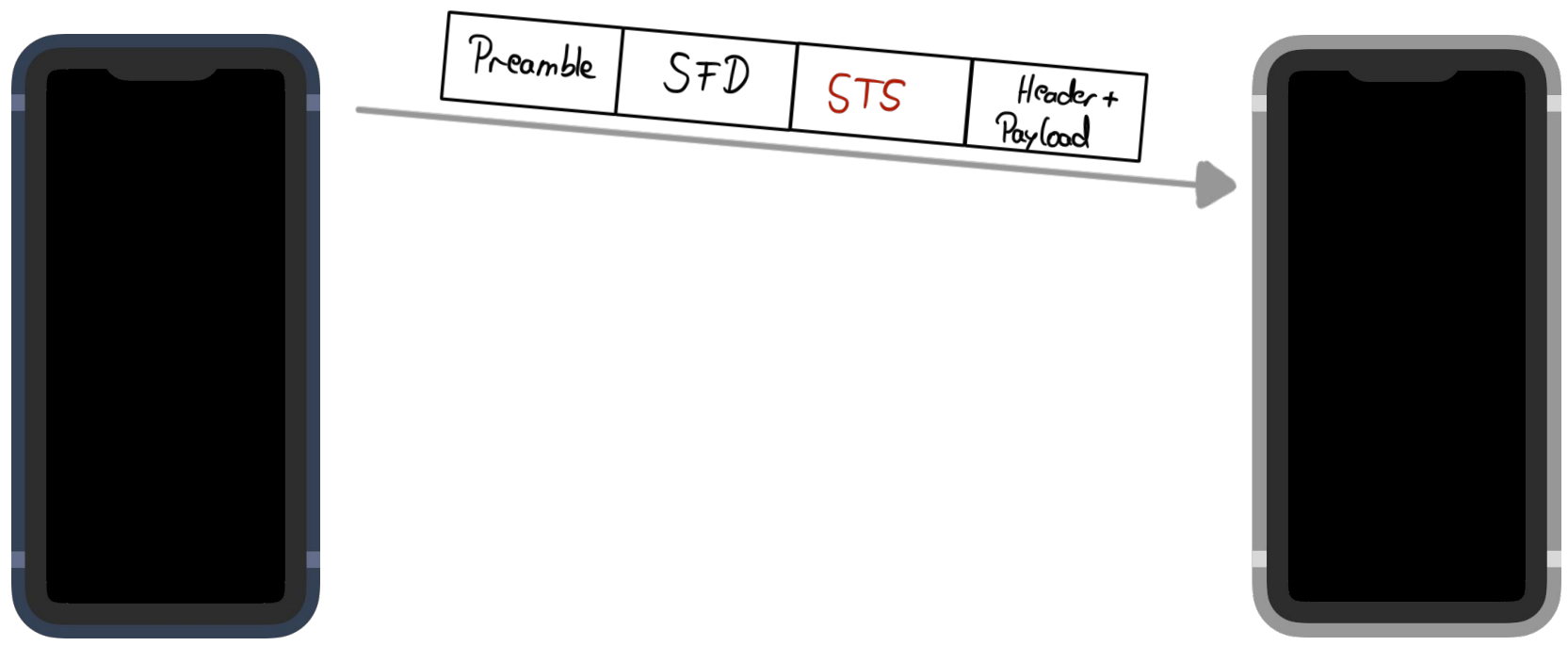
NIDiscoveryToken



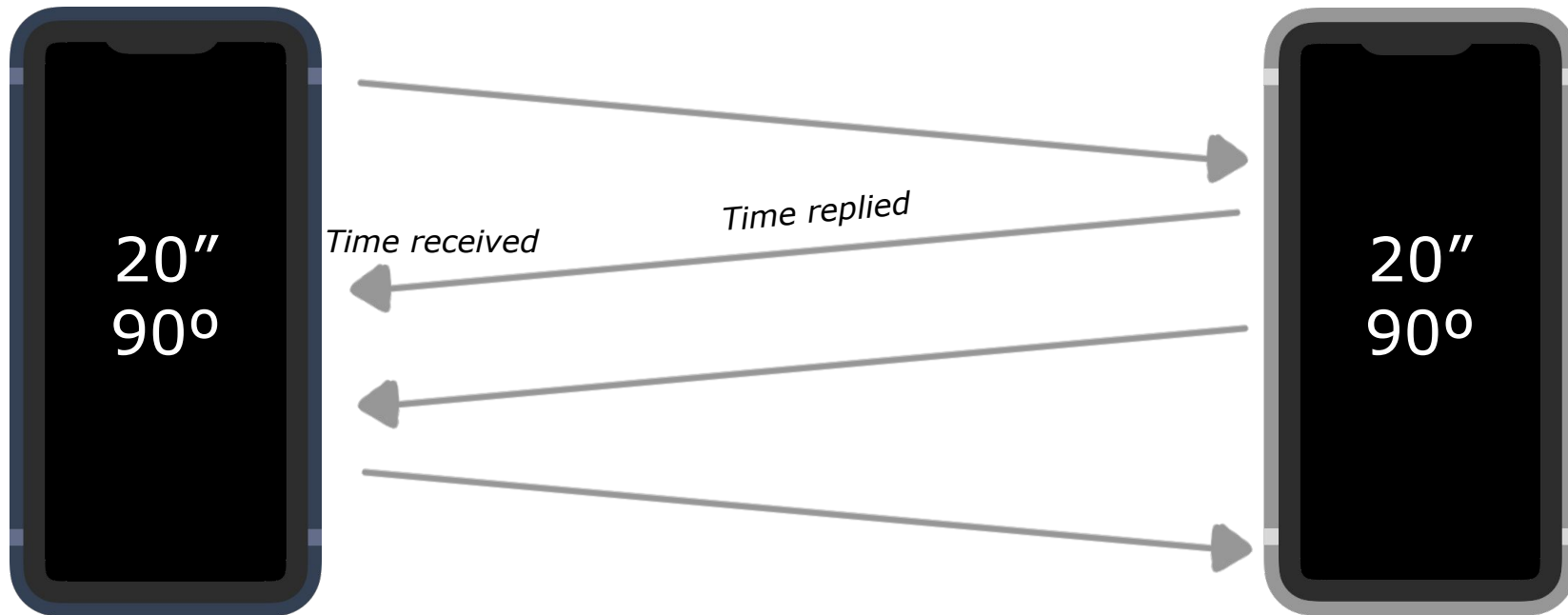
$\text{SipHash}(\text{MAC address}, \text{IRK}) = \text{Auth Tag}$

UWB Secure Ranging

^
Somewhat



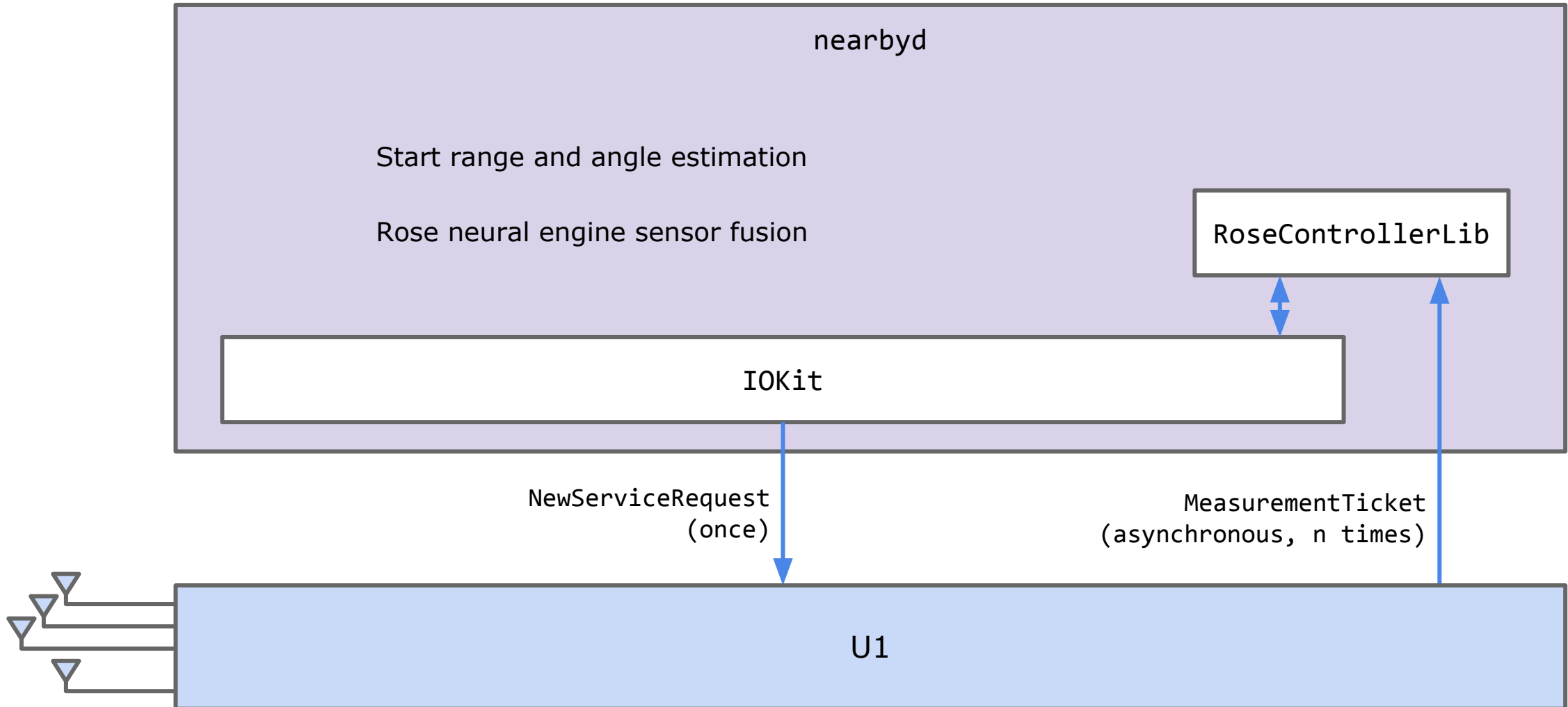
UWB Ranging



Time of flight = Time received - Time replied - processing time

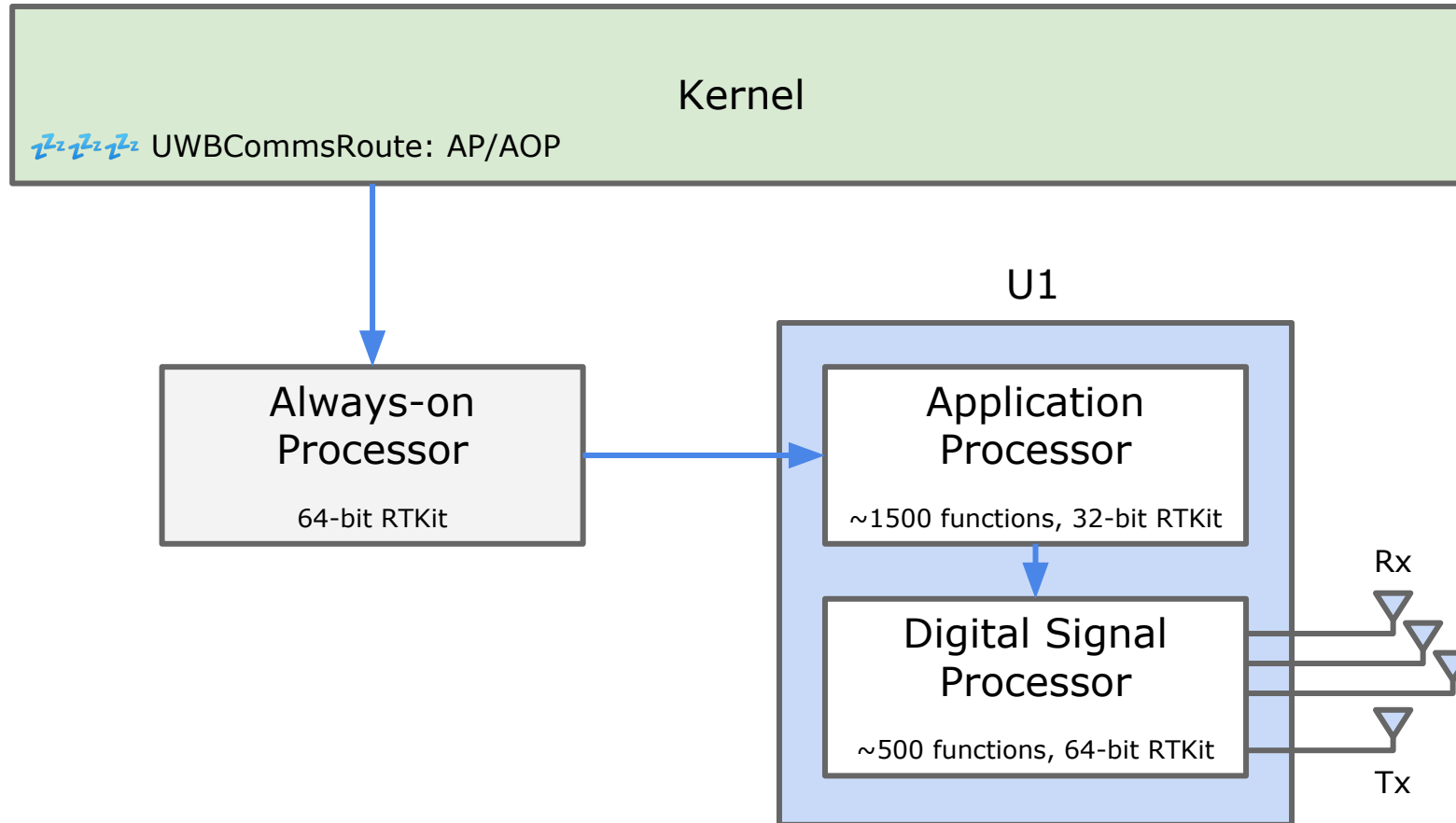
AirDrop	Nearby Interaction
One-to-many ranging	Peer-to-peer ranging
Single sided ranging	Double sided ranging
Likely no STS	Shared secret and STS

AoA and Distance Measurement Ticket Processing

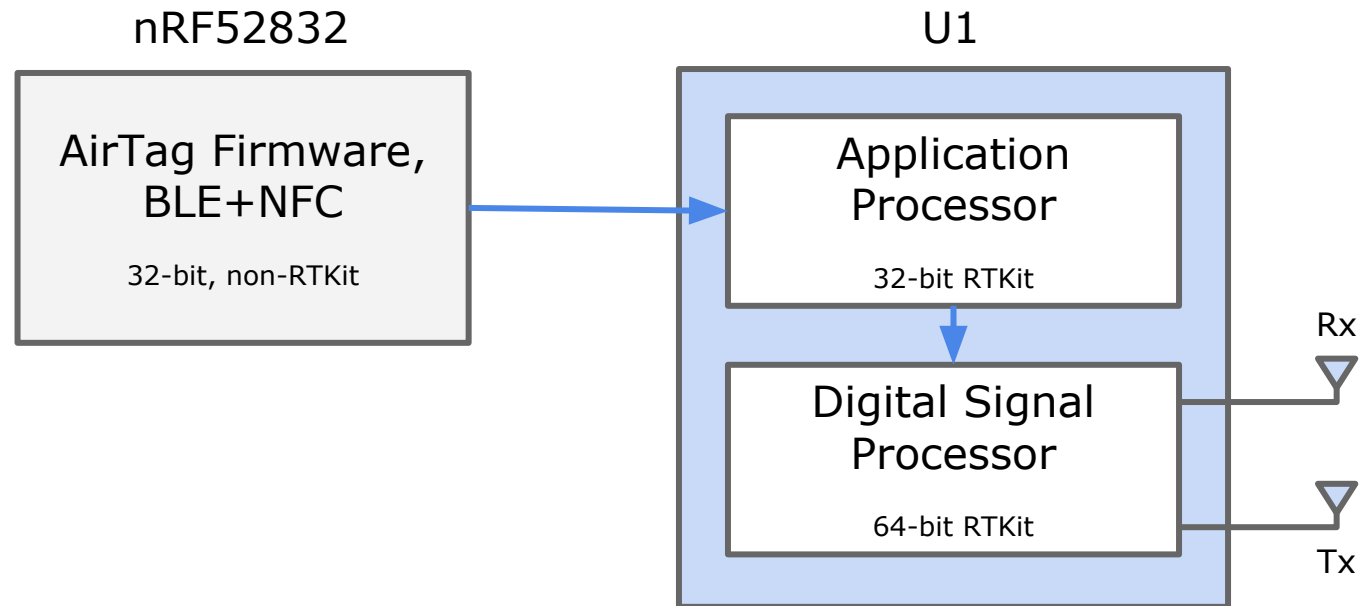


Hardware Interaction


Hardware Components



Hardware Components - AirTag

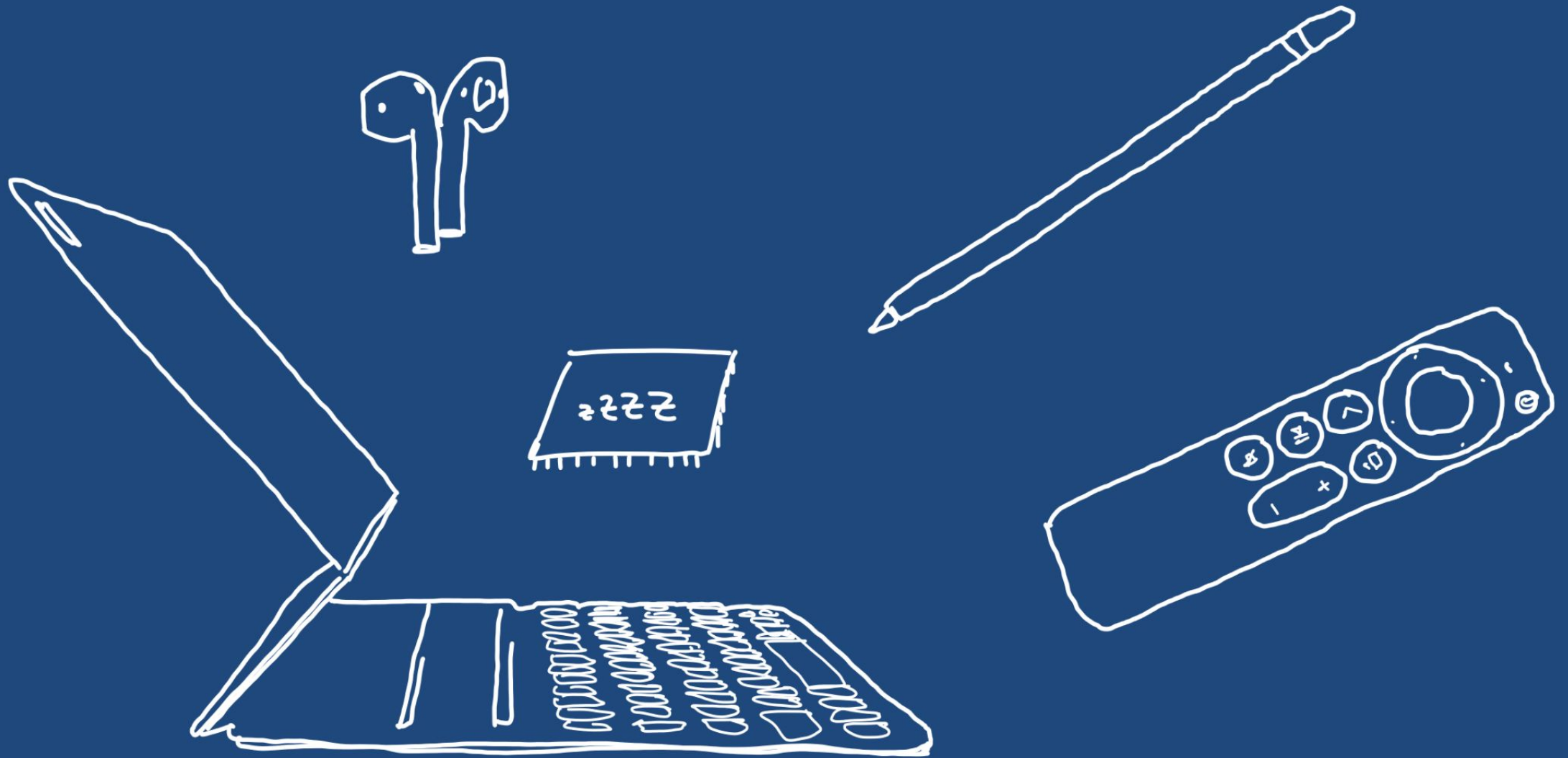


RTKit Operating System

- RTKitOS runs on almost every Apple chip or embedded device.
 - 64-bit variant comes with ASLR. 
 - Lightweight, ~100 functions.
 - Even logging is implemented differently in every RTKitOS firmware.
- RTKitOS debug builds support additional logging.
 - U1 debug builds: iOS 13.3 on iPhone 11 & initial AirTag firmware 🎉

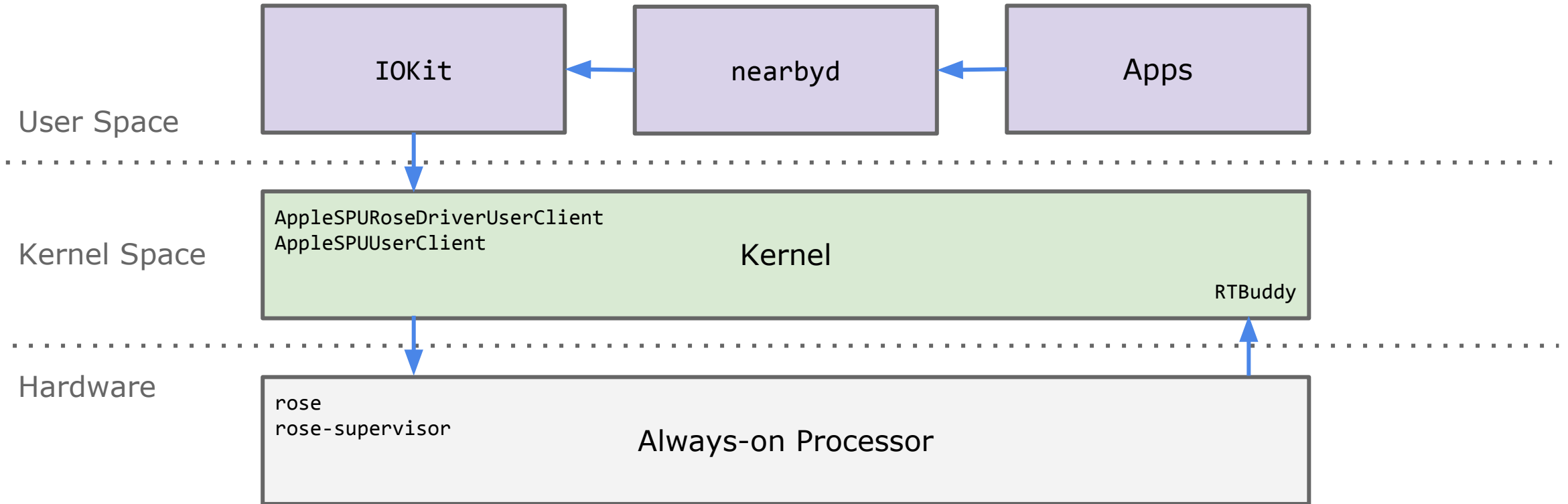
It's bigger on
the inside!





More details about RTKitOS in Apple's Bluetooth chip and peripherals are documented in Dennis Heinze's thesis (<https://github.com/seemoo-lab/toothpicker>).

Duplicate User Clients



IOKit UserClients for RTKit-based chips have equivalents in the AOP. Same principle for other wireless chips by Apple, e.g., the audioOS AOP implements `marconi-bluetooth` and `aop-marconi-bt-control` to communicate with Apple's Bluetooth chip.

RTKit-based chips communicate with an RTBuddy for logging etc.

Checking RTKit-based Driver Dependencies

```
# ioreg -rtc IOUserClient

+-o Root <class IORegistryEntry, id 0x100000100, retain 184>
  +-o N104AP <class IOPlatformExpertDevice, id 0x10000020f, ... >
    +-o AppleARMPE <class AppleARMPE, id 0x100000210, ... >
      +-o arm-io@10F00000 <class IOPlatformDevice, id 0x100000118, ... >
        ...
          +-o RTBuddyV2 <class RTBuddyV2, id 0x100000374, ... >
            +-o AOPEndpoint17 <class RTBuddyEndpointService, id 0x1000003a0, ... >
              +-o AppleSPU@10000014 <class AppleSPU, id 0x1000003dc, ... >
                +-o rose <class AppleSPUAppInterface, id 0x100000142, ... >
                  +-o AppleSPURoseDriver <class AppleSPURoseDriver, id 0x1000004e4... >
                    +-o AppleSPURoseDriverUserClient <class AppleSPURoseDriverUserClient, id 0x100000aa3, ... >
                      {
                        "IOUserClientCreator" = "pid 549, nearby"
                      }
                ...
              +-o AppleSPU@10000020 <class AppleSPU, id 0x1000003e2, ... >
                +-o rose-supervisor <class AppleSPUHIDInterface, id 0x10000049e, ... >
                  +-o AppleSPUUserClient <class AppleSPUUserClient, id 0x100000aa4, ... >
                    {
                      "IOUserClientCreator" = "pid 549, nearby"
                      "IOUserClientDefaultLocking" = Yes
                    }
                ...
              ...
            ...
          ...
        ...
      ...
    ...
  ...
```

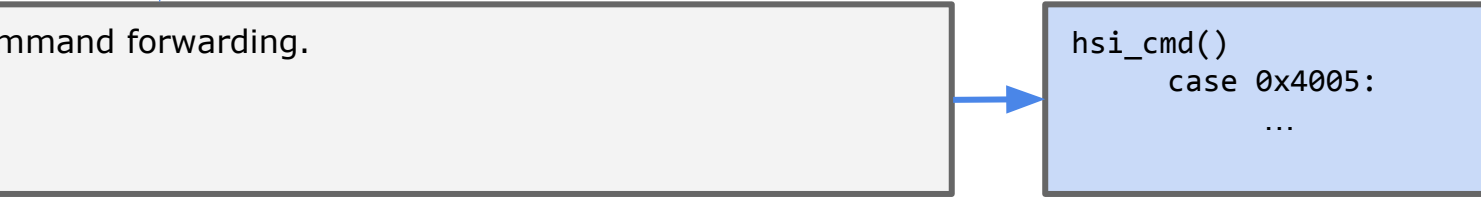
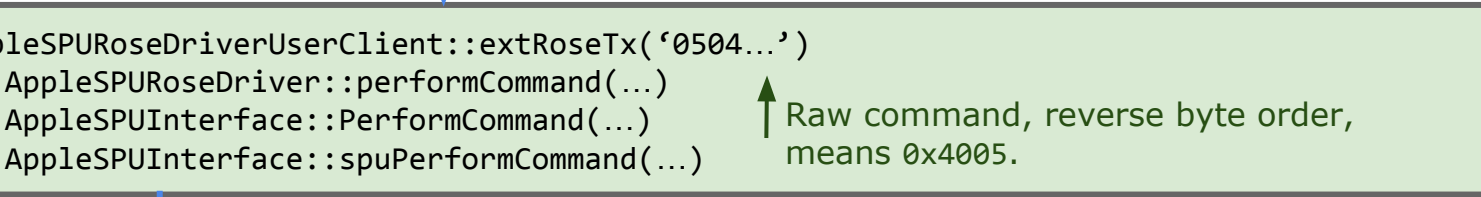
Find detailed ioreg outputs from current devices on <https://github.com/robre/ioreg-archive>.

Sending Commands directly to Rose

User Space



- 0 extRoseLoadFirmware
- 1 extRoseGetInfo
- 2 extRoseReset
- 3 extRoseEnterCommandMode
- 4 extRosePing
- 5 extRoseTx
- 6 extRoseTimeSync
- 7 extRoseGetSyncedTime
- 8 extRoseGetProperty
- 9 extRoseSetProperty
- 10 extRosePerformInternalCommand
- 11 extRoseCacheFirmwareLogs
- 12 extRoseDequeueFirmwareLogs
- 13 extRoseTriggerCoredump
- 14 extRoseDequeueCoredump
- 15 extRoseCoredumpInfo
- 16 extRosePowerOn
- 17 extRoseReadPowerState
- 18 extRoseConfigureFirmwareLogCache

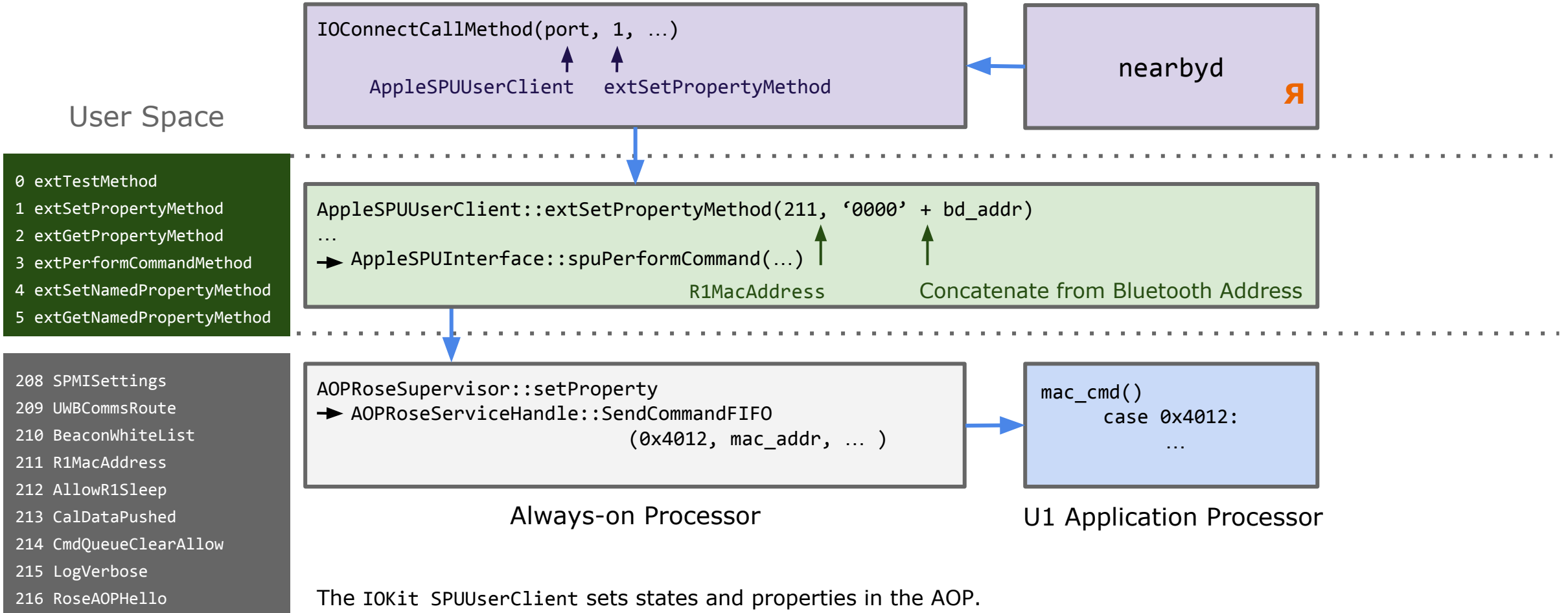


Always-on Processor

U1 Application Processor

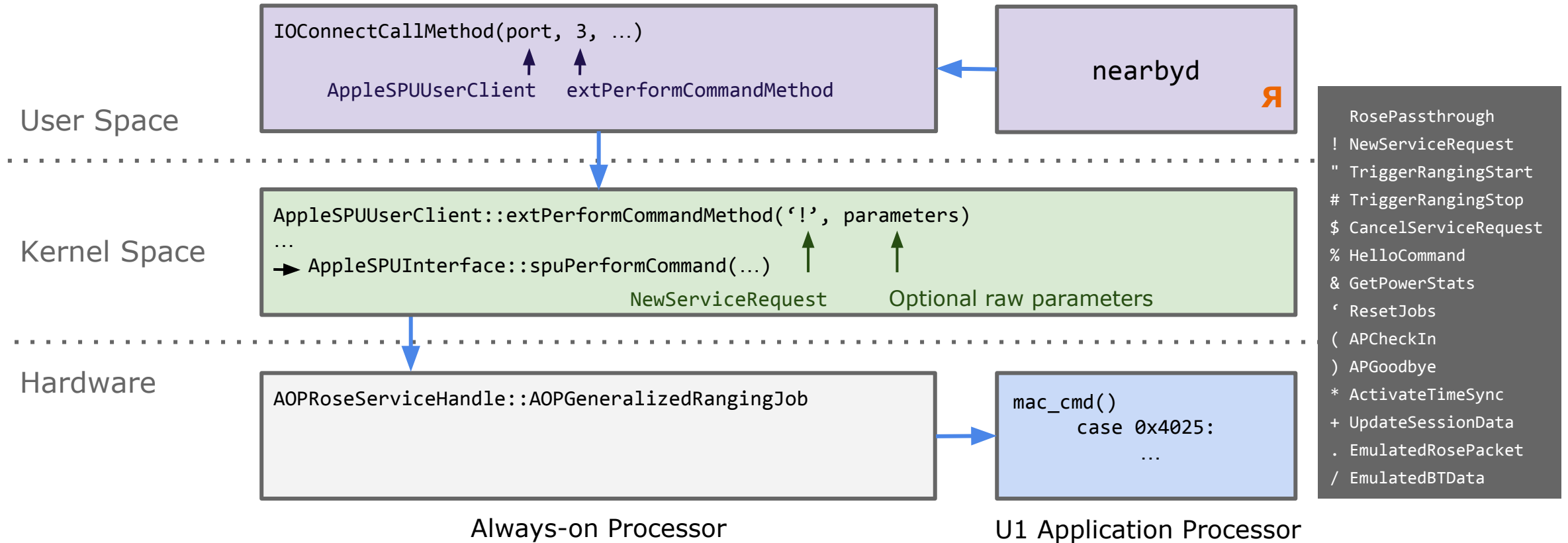
The IOKit RoseDriverUserClient exports various functions, but in the end they call AppleSPUIterface::spuPerformCommand(...) within the kernel, similar to the SPUUserClient.

Sending Commands via the AOP to Rose



The IOKit SPUUserClient sets states and properties in the AOP. If needed, certain state changes also apply commands to the U1 chip.

Sending Commands via the AOP to Rose



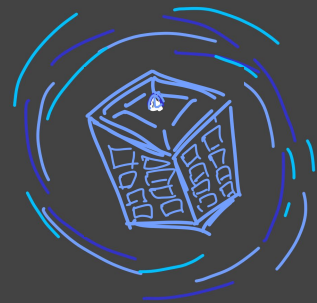
Demo: Frida script that decodes interaction
-> NewServiceRequest etc.

GR Packet to Initiate Secure Ranging

```

nearbyd[1184] <Notice>: RoseScheduler::handleNewServiceRequestInternal
nearbyd[1184] <Notice>: [AP Scheduler] Servicing dequeued service request.
    Passing message to AOP scheduler.
nearbyd[1184] <Notice>: Request: [Role]: Initiator, [MacMode]: GR
nearbyd[1184] <Notice>: Built GR packet: {
    ses_role: 0
    , tx_ant_mask : 2
    , rx_ant_mask : 11
    , rx_sync_search_ant_mask : 2
    , tx_preamble: 3
    , rx_preamble: 3
    , tx_pkt_type: 0
    , rx_pkt_type: 0
    , tx_mslot_sz_250us: 12
    , rx_mslot_sz_250us: 12
    , interval_min_ms: 30
    , naccess_slots_min: 1
    , naccess_slots_max: 32
    , access_slot_idx: 0
    , start_channel: 1
    , alternate_channel: 0
    , channel_hop_pattern_mask: 8
    , debug_flags: 7
    , start_time: 0
    , start_time_uncertainty: 0
    , interval_max_ms: 5000
    , local_addr: 0x0
    , peer_addr: 0x0
    , sts_blob: 1281711291571851042031941281011261981431306684
}

```



Scrambled Timestamp Sequence

0x80=128, 0xab=171, ..., 0x54=84

```

- AppleSPUUserClient::extPerformCommandMethod()
  > connection      0xa503
  > selector        0x3
  > input
    v
      0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
00000000  21                                     !
+ NewServiceRequest
  v---- IOKit input struct ----
      0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
00000000  30 00 16 00 00 00 04 00 01 13 01 02 00 00 00 00 0.....
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000d0  00 00 00 00 25 40 00 00 00 00 01 04 02 0b 02 01 ...%@.....
000000e0  00 08 03 03 00 00 00 00 00 00 0c 0c 00 00 1e 00 .....
000000f0  88 13 01 20 ff 80 ab 81 9d b9 68 cb c2 80 65 7e ... ..h...e~
00000100  c6 8f 82 42 54 00 00 00 00 00 07 00 00 00 00 00 ...BT.....
00000110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000120  00 00 00 00 00 00 00 00

```

Firmware Format

U1 Firmware Extraction

Contained in every iOS/audioOS IPSW, watchOS OTA image, or AirTag firmware image.

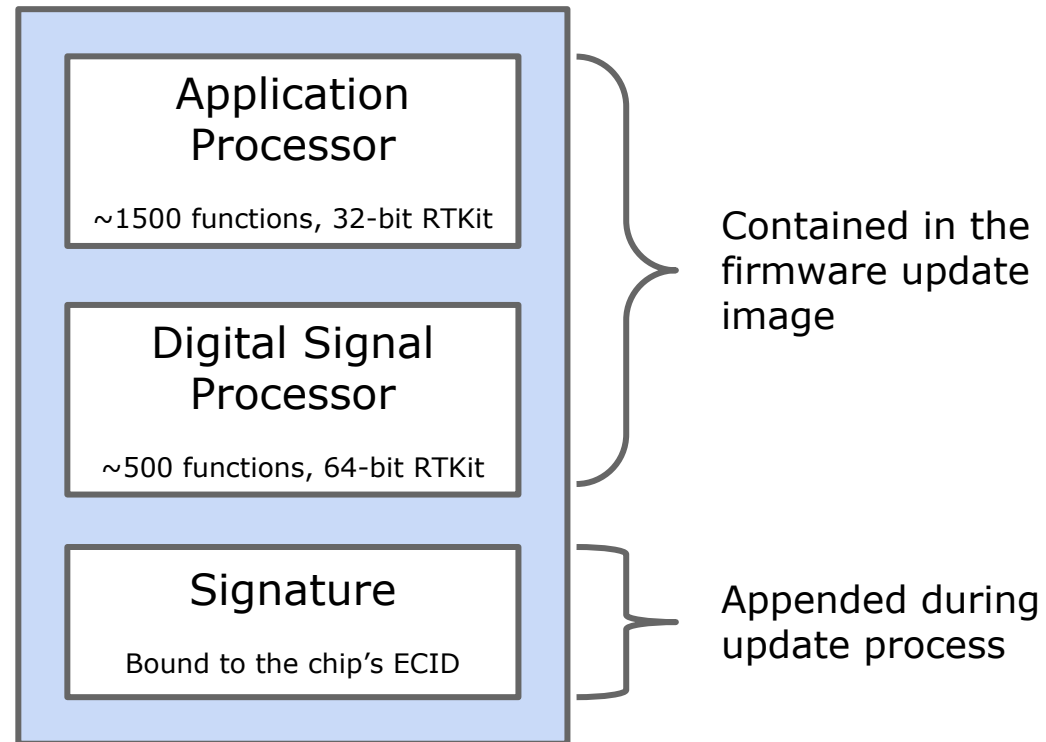
`/Firmware/Rose/[type]/ftab.bin`

Types as of now:

- iPhone 11 (r1p0)
- iPhone 12 (r1p1)
- Apple Watch 6 (r1w0)
- HomePod mini (r1hp0)
- AirTag (b389)

```
00000000 01 00 00 00 ff ff ff ff 00 00 00 00 00 00 00 .....
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000020 72 6b 6f 73 66 74 61 62 03 00 00 00 00 00 00 rkosftab.....
00000030 72 6b 6f 73 60 00 00 00 e0 98 04 00 00 00 00 rkos`.....
00000040 73 62 64 31 40 99 04 00 60 39 04 00 00 00 00 sbd1@...`9.....
00000050 62 76 65 72 a0 d2 08 00 26 00 00 00 00 00 00 bver....&.....
...
000786b0 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 .....
000786c0 52 54 4b 69 74 5f 69 4f 53 2d 31 32 36 34 2e 36 RTKit_iOS-1264.6
000786d0 30 2e 36 2e 30 2e 31 2e 64 65 62 75 67 00 00 00 0.6.0.1.debug...
000786e0 06 00 00 80 04 00 00 00 00 00 00 00 00 00 00 .....
000786f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```


Firmware Segments



Save BLOBs!

Demo: Show system messages how the chip boots, maybe also an invalid boot

Obtaining Logs

Trigger Rose Error Handling (#1)

Can we interact with the firmware without modifying it?

SystemOff is executed when entering flight mode, switch this with the implementation of TriggerFatalErrorHandling.

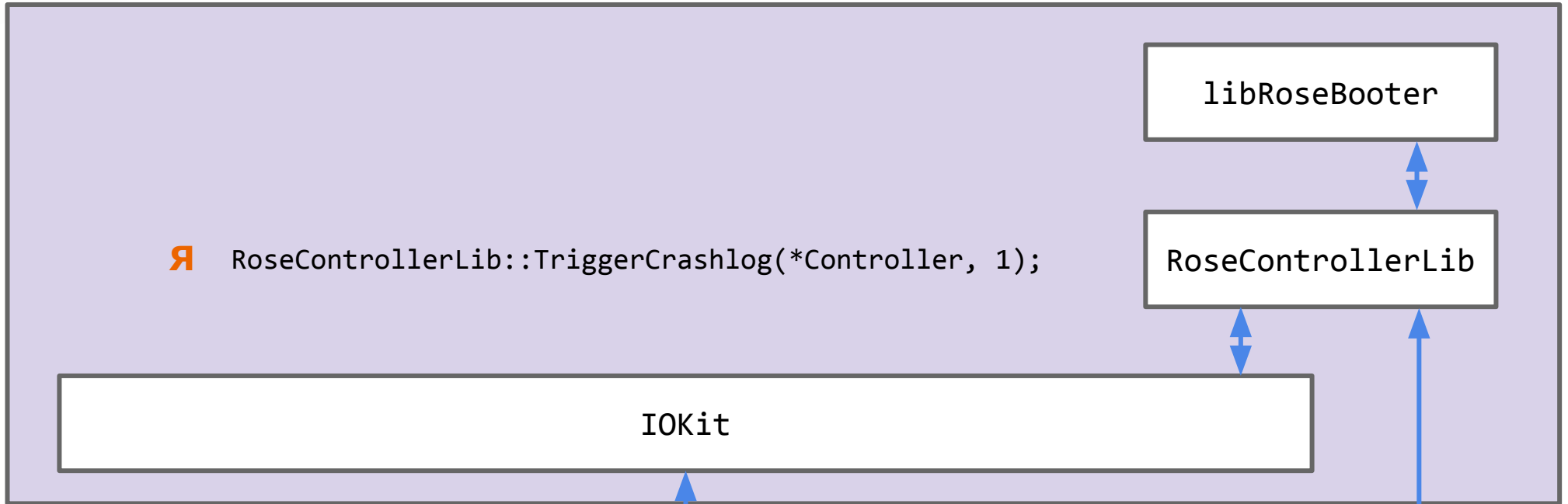
Get full crash logs and packet logs by setting isInternalBuild and a few other properties.

```
case 7:
    _os_log_impl( ...,
        "PRRoseProvider::relayCommandMessage -- SystemOff",
        buf_full_packet,
        2LL);
    ...
case 8:
    ...
    "PRRoseProvider::relayCommandMessage -- RefreshConfiguration",
    ...
    PRRoseProvider::relayCommandMessage_RefreshConfiguration_104F70484(a1 + 19);
    ...
case 9:
    ...
    "PRRoseProvider::relayCommandMessage -- TriggerFatalErrorHandling",
    ...
    log_rose_r1_msg_1021139CC(buf_full_packet, "AOPRoseFatalError");
    PRRoseProvider::relayCommandMessage_TriggerFatalErrorHandling_104F72654(...)
    ...
```

Trigger Rose Error Handling (#2)

Shared libraries export symbols!

nearbyd



```
Я RoseControllerLib::TriggerCrashlog(*Controller, 1);
```

libRoseBooter

RoseControllerLib

IOKit

User Space

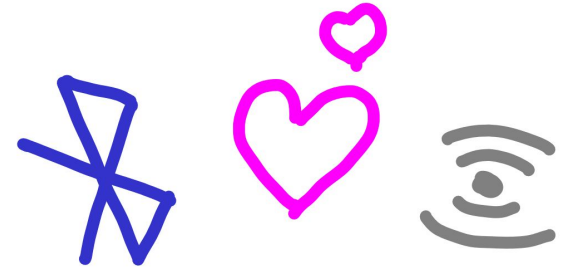
Mach Messages

Kernel Space

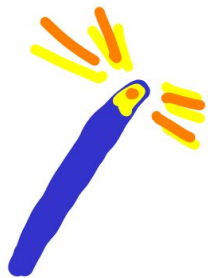
Demo: Show Crash Logs & iOS 13.3 Packet Logs

Conclusion

Lessons Learned



- Bluetooth and Ultra Wideband are tightly coupled on iOS.
- Apple's own RTKit-based wireless chips have an interesting architecture with many security features like secure boot and ASLR.
- Many features in the chip can be instrumented from user space.



Magie!

Q&A

 <https://github.com/seemoo-lab>

 Twitter: @naehrdine, @Sn0wfreeze

 [\[jclassen|aheinrich\]@seemoo.de](mailto:[jclassen|aheinrich]@seemoo.de)