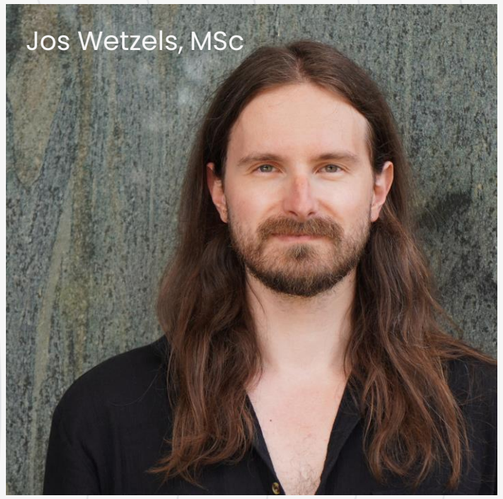
MIDNIGHT
BLUE

November 2023

All Cops Are Broadcasting

Obtaining the secret TETRA primitives after decades in the shadows

By Midnight Blue



Midnight Blue

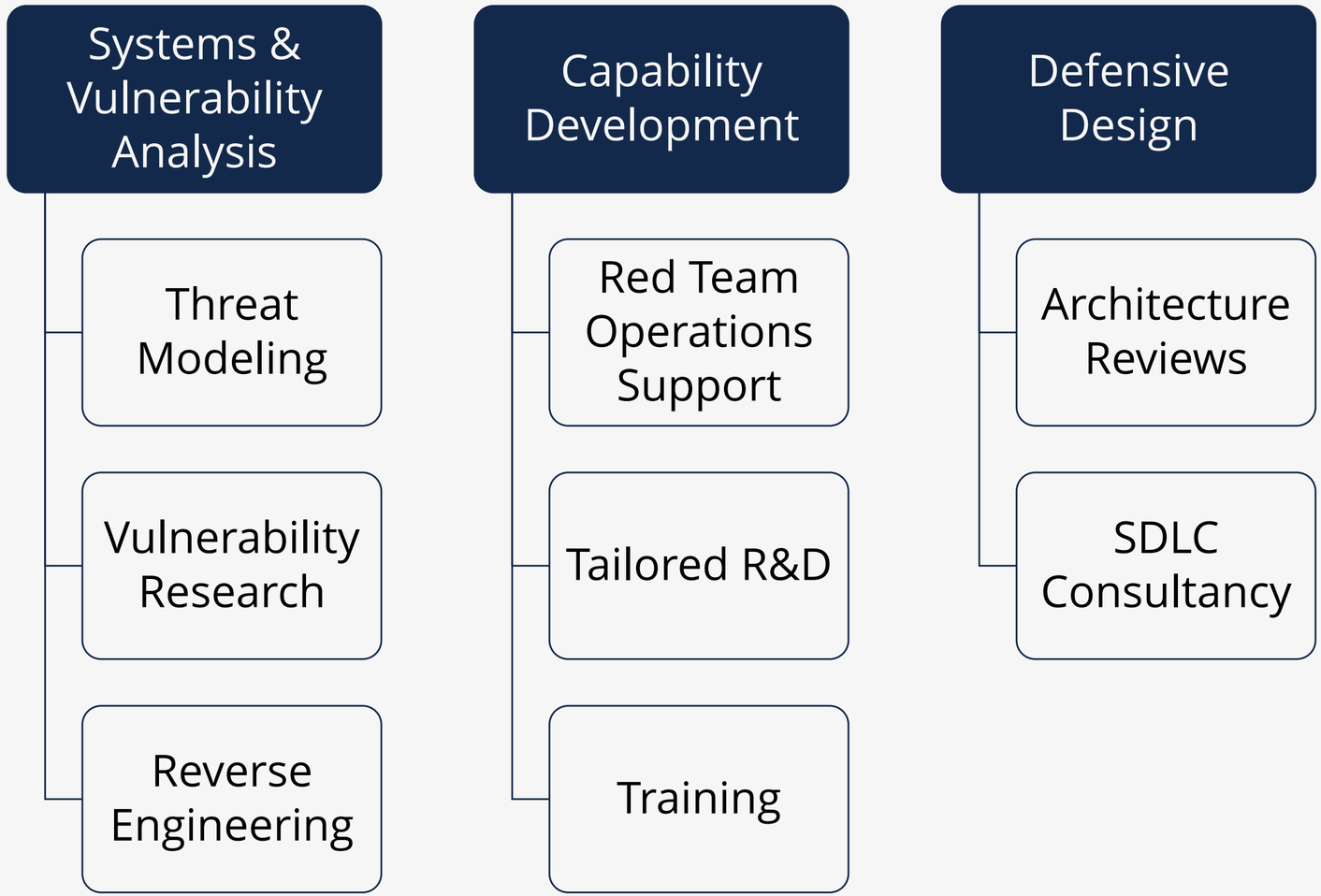
FCA
PSA

BlackBerry®
QNX®

MIFARE Classic

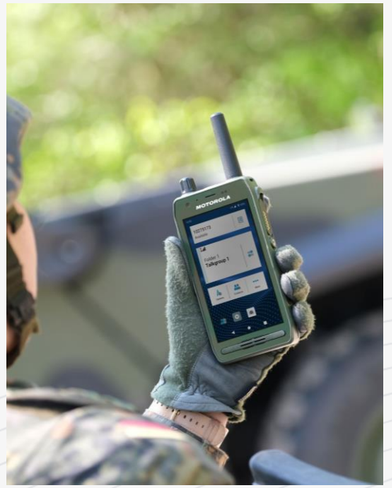
Selected Research

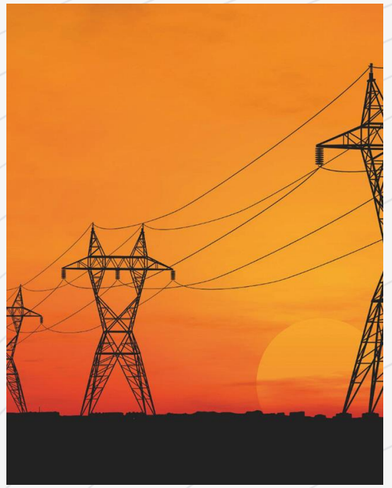
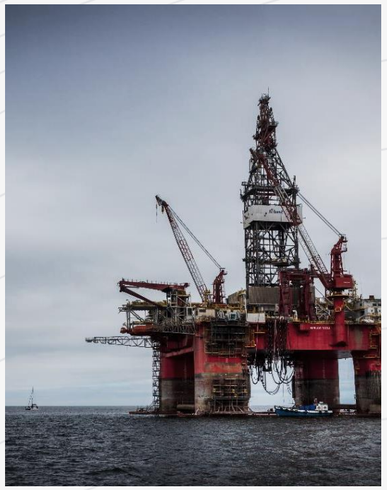
Services



What is TETRA?

- Globally used radio technology
  - Competes with P25, DMR, TETRAPOL
- Standardized in 1995 by ETSI
  - Known for GSM, 3G/4G/5G, GMR, etc.
- Used for voice & data communications incl. machine-to-machine
- Relies on secret, proprietary cryptography

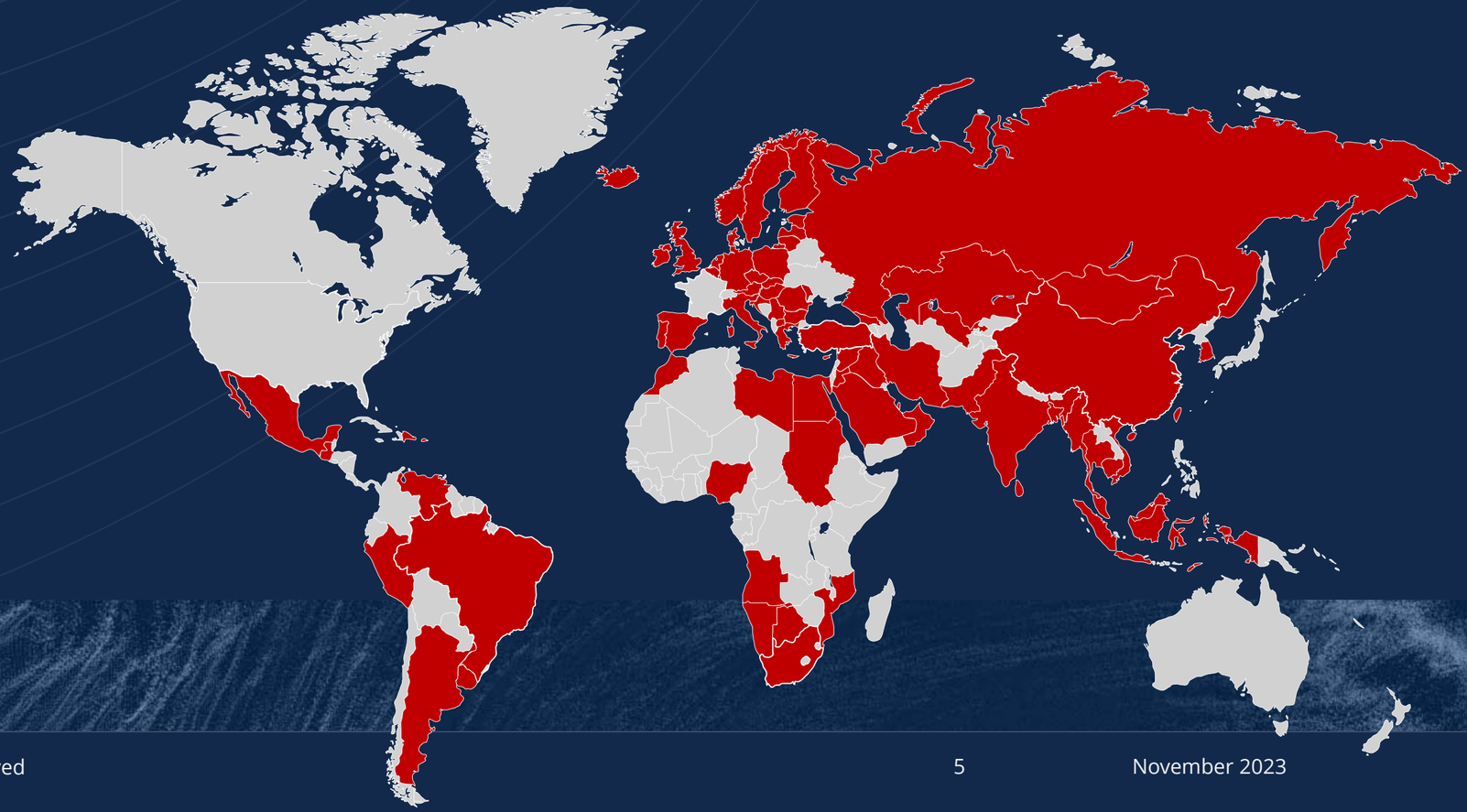


Use by police

Vast majority of global police forces use TETRA radio technology.

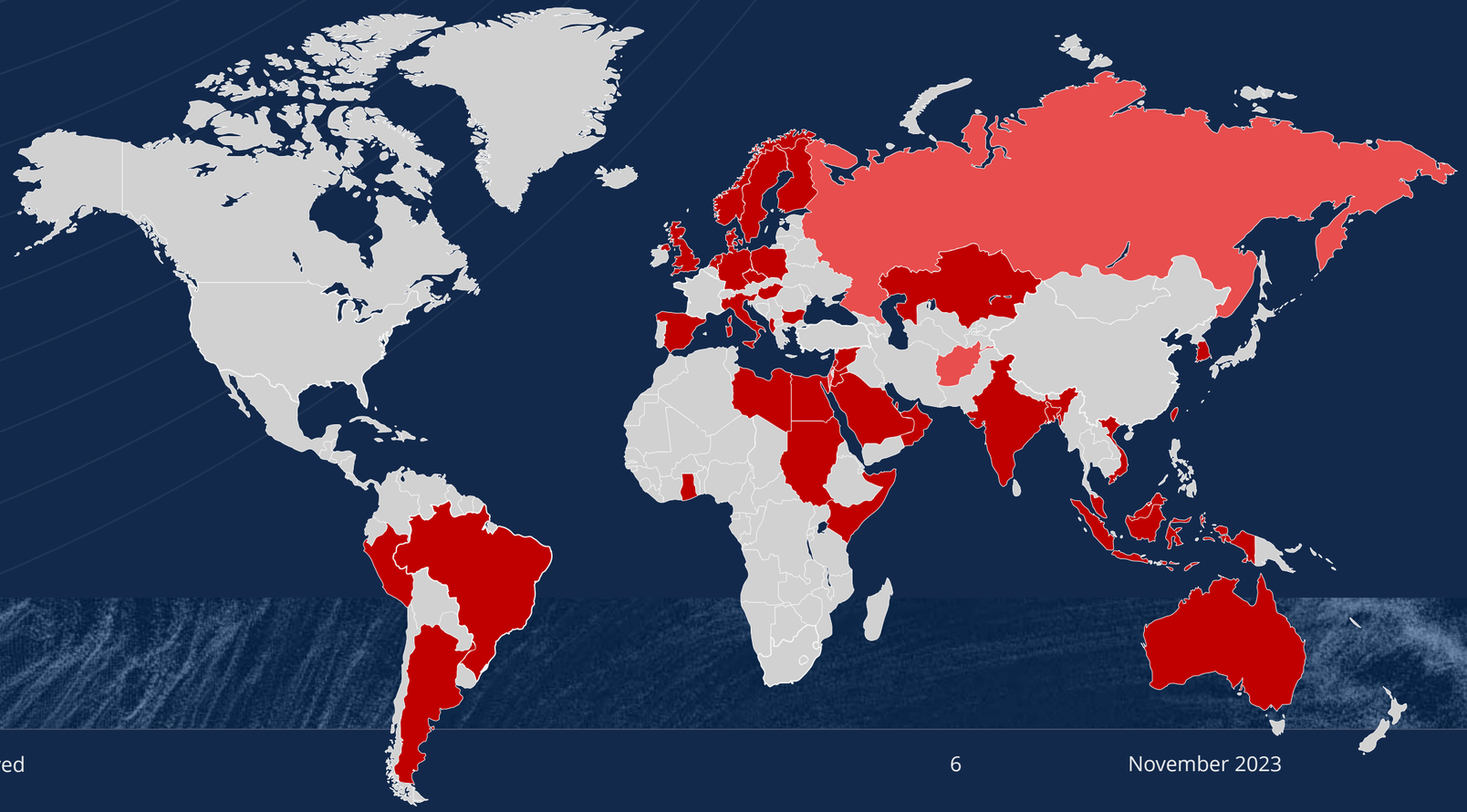
- C2000 (NL)
- ASTRID (BE)
- BOSNET (DE)
- AIRWAVE (UK)
- Nødnett (NO)
- Rakel (SE)
- SINE (DK)
- VIRVE (FI)
- SIRESP (PT)
- …



Based on OSINT

Military & Intelligence

Many countries have one or more military or intelligence units using TETRA radio technology as primary, fallback, or interfacing comms.

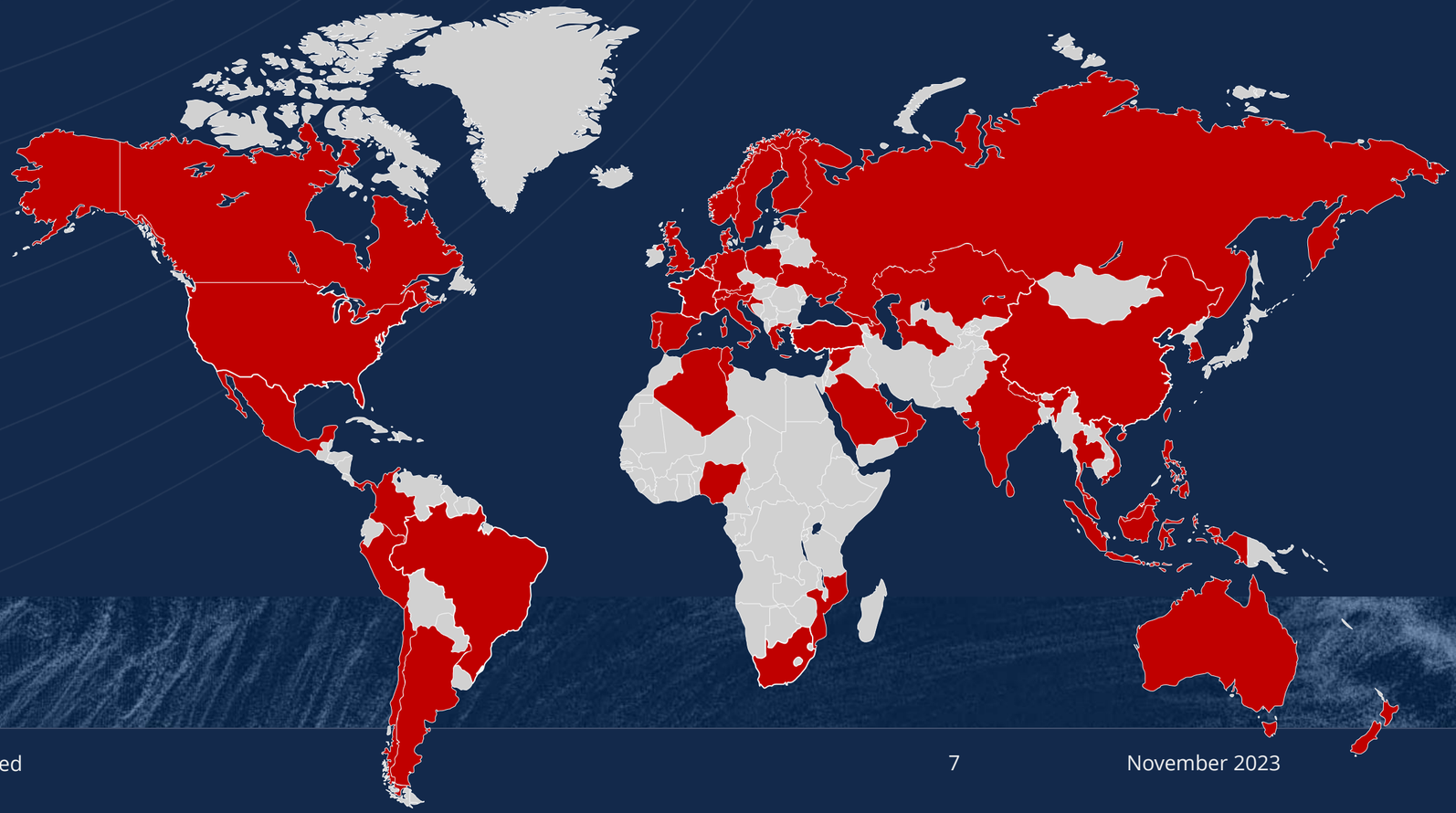


Based on OSINT

Critical Infrastructure

Many parties such as airports, harbors, and train stations use TETRA for voice communications.

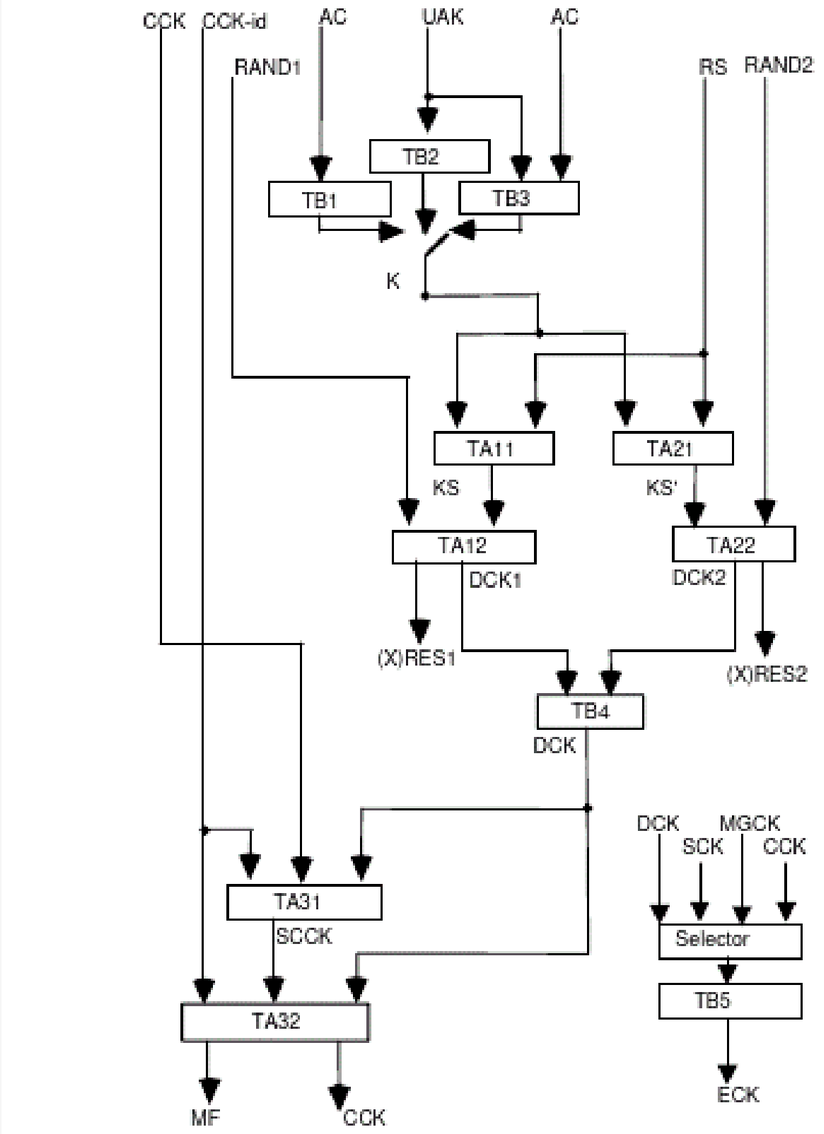
In addition TETRA is used for SCADA WAN, such as substation & pipeline control, or railway signalling.



Based on OSINT

Open standard?

- Public standard, secret crypto
  - NDAs, only available for ‘bona fide’ parties
- Manufacturers must protect algorithms
  - Hardware, or, implementations
  - Software with extraction countermeasures
- Top-tier adversaries likely have specs (e.g. via in-country manufacturers or theft)
- We probably had to jump through more hoops…



ıre B.1: Overview of air interface authentication and key management (sheet 1

TETRA security

- **TAA1 suite**
  - Authentication, key management / distribution (OTAR)
  - Identity encryption
  - Remote disable

- **TEA (TETRA Encryption Algorithm) suite**
  - Voice and data encryption (Air Interface Encryption (AIE))
    - **TEA1: Readily exportable**
    - **TEA2: European public safety**
    - **TEA3: Extra-European public safety**
    - **TEA4: Readily exportable (hardly used)**
  - Not to be confused with Tiny Encryption Algorithm!

- **Optional end-to-end encryption**

Project RE:TETRA

Kerckhoffs' principle

"A cryptosystem should be secure even if everything about the system, except the key, is public knowledge."

-Auguste Kerckhoffs, 1883

Violators don't fare well

- A5/1, A5/2 (GSM), COMP128 (GSM)
- GMR-1, GMR-2 (SATPHONES)
- GEA-1, GEA-2 (GPRS)
- DSAA, DSC (DECT)
- MIFARE (RFID)
- HITAG (RFID)
- MEGAMOS (RFID)
- DST (RFID)
- Legic (RFID)
- CSS (DVD)
- CryptoAG / Hagelin

- Orange = backdoored

~~Kerckhoffs' principle~~ ETSI's principle

"Well [obscurity is] also a way of maintaining security."*

-Brian Murgatroyd, Chairman ETSI TC TETRA, 2023

* Interview between Kim Zetter and Brian Murgatroyd, Chair of ETSI TC TETRA
https://zetter.substack.com/p/interview-with-the-etsi-standards

Project motivation

- Proprietary cryptography has repeatedly suffered from practically exploitable flaws which remain unaddressed until disclosed

- GOAL: open up TETRA for public review after 20+ years
  - Enables informed risk analysis
  - Resolve issues
  - Level playing field

- Funded by NLnet
  - NPO funding open IT projects

Let’s break open a radio!

Selecting a target

- **Wide variety of:**
  - Brands
  - Models
  - Architectures
- **We need to pick well**
  - Wrong choice wastes lot of time

Vendor	Baseband/fam	Stds	IC	Arch MCU/DSP	Sec
Motorola	Patriot	DMR P25	DSP566xx	MCORE / DSP56600	HW
Motorola	Redcap2	-	DSP566xx	MCORE / DSP56652	HW
Motorola	Freon	DMR P25	OMAP-L138	ARM / TMS320C674x	TEE
Hytera	-	DMR	OMAP5912ZZG	ARM / TMS320C55x	FPGA
Sepura	GEN3 CastleCombe	-	OMAP-L138	ARM / TMS320C674x	HW
Leonardo	Leonardo / SELEX	-	OMAP 3730	ARM / TMS320C64x	?

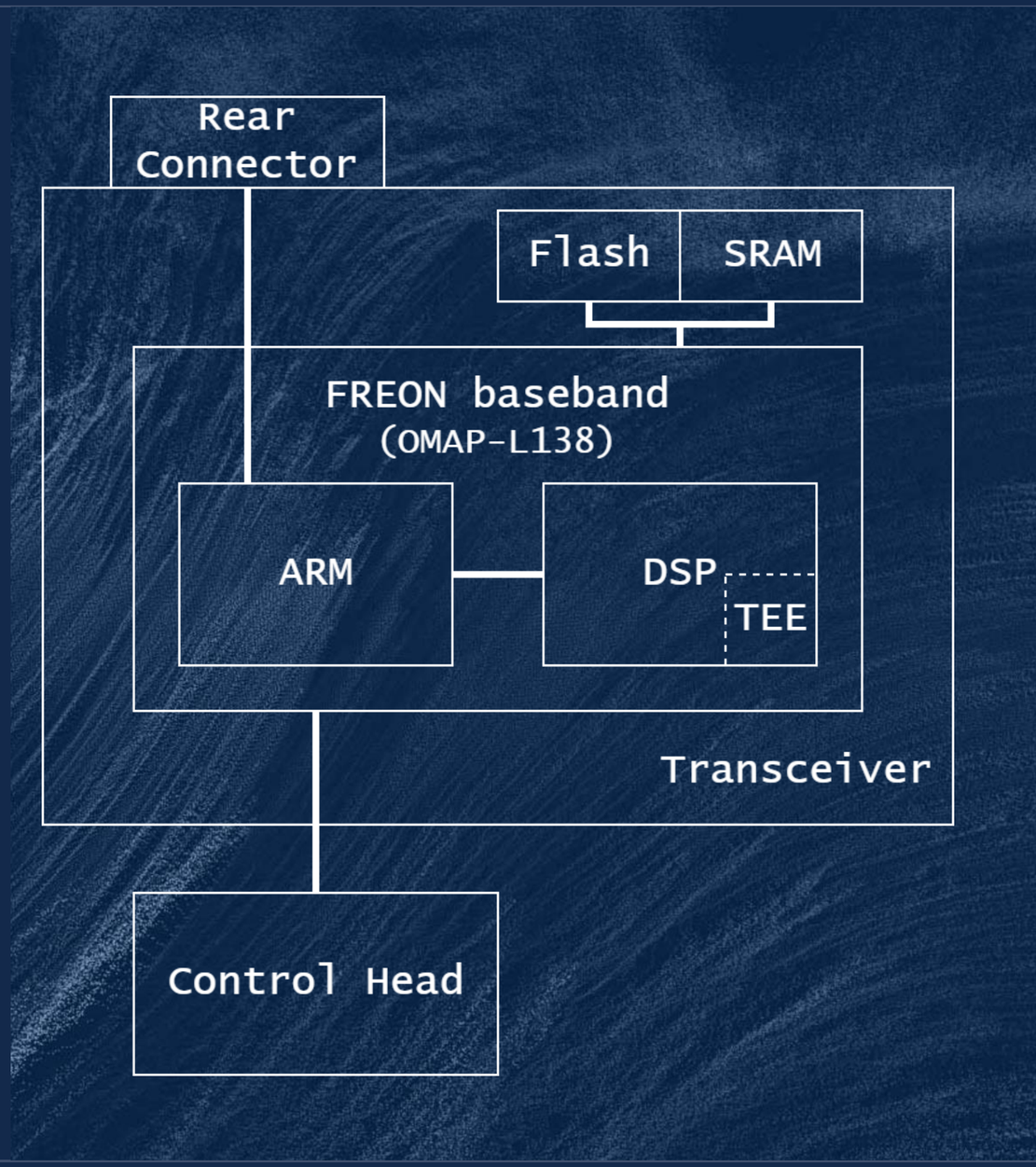
Vendor	Baseband/fam	Stds	IC	Arch MCU/DSP	Sec
Motorola	Patriot	DMR P25	DSP566xx	MCORE / DSP56600	HW
Motorola	Redcap2	-	DSP566xx	MCORE / DSP56652	HW
Motorola	Freon	DMR P25	OMAP-L138	ARM / TMS320C674x	TEE
Hytera	-	DMR	OMAP5912ZZG	ARM / TMS320C55x	FPGA
Sepura	GEN3 CastleCombe	-	OMAP-L138	ARM / TMS320C674x	HW
Leonardo	Leonardo / SELEX	-	OMAP 3730	ARM / TMS320C64x	?

Motorola MTM5400

- Common model, easily obtained 2nd hand online
- Baseband SoC by TI
  - So, no hardware TETRA crypto
- SoC has software security features
  - Used for protecting TETRA crypto from extraction?

Architecture

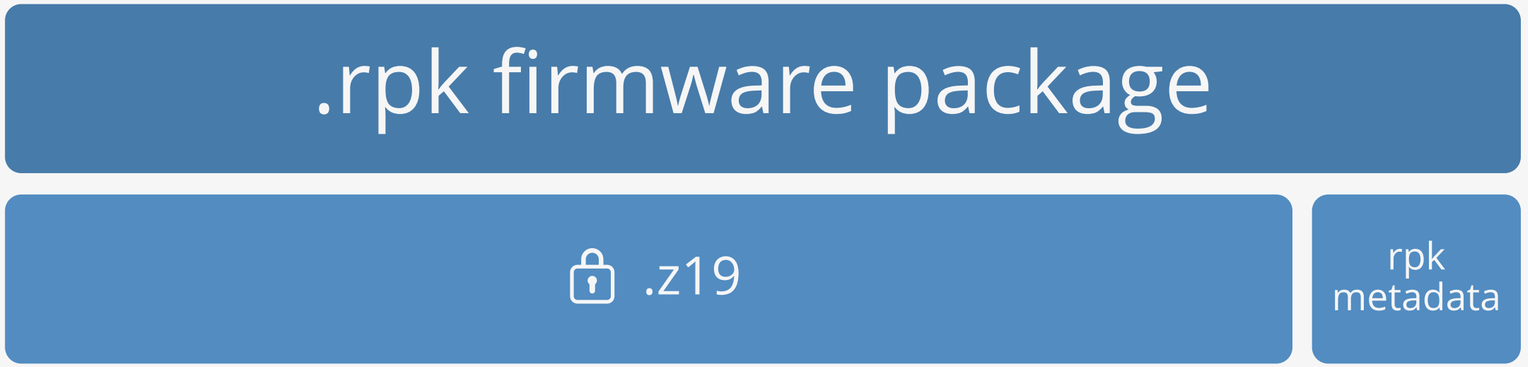
- **Control head**
  - Microphone, keyboard, field configuration
- **Rear connector**
  - Depot programming
- **Baseband**
  - ARM core (high-level stuff)
  - DSP (signals processing)
  - TEE (magic sauce?)



MTM Firmware format

- Shipped in .rpk package

- But how to extract it?



- Zip archive
- Encrypted..

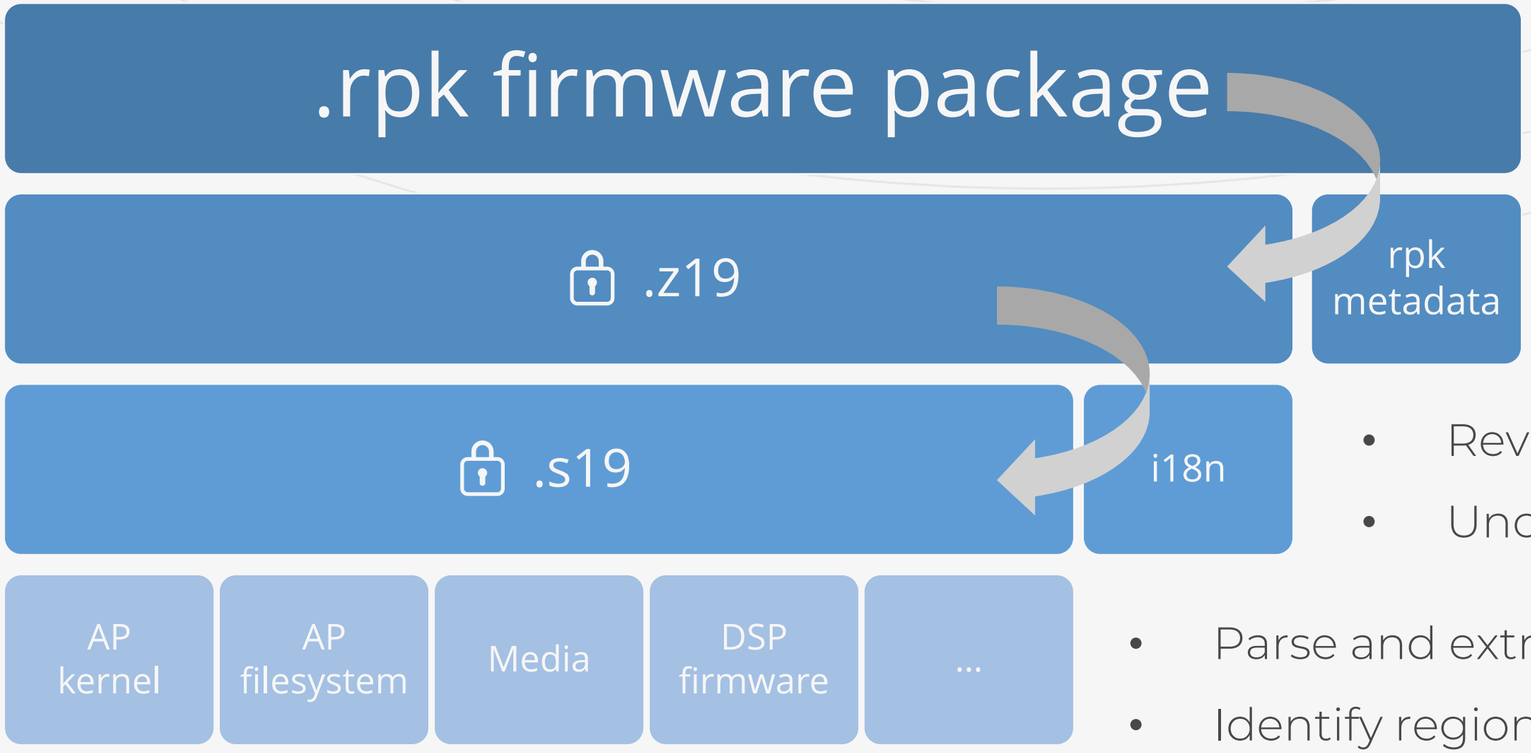
Warmup

- Firmwares are not personalized
  - Has to be a general way to load them
- TetraCPS decrypts files in .rpk
  - … and then files inside that, yet again
- Some light .NET reversing

```
label_16:
            try
            {
              string str;
              switch (0)
              {
                case 0:
label_18:
                  A_0.Password = "[redacted]";
                  str = Path.GetFileName(this.h).ToLowerInvariant();
                  num2 = 5;
                  num1 = num2;
                  goto default;
                default:
                  rv rv;
                  ZipEntry entry;
                  while (true)
                  {
                    switch (num1)
                    {
```

```
public void Open(string path, string tempRootFolder)
{
  Guard.ArgumentNotNullOrEmpty(path, nameof (path));
  this.FilePath = path;
  if (!File.Exists(path))
  {
    this.zipCreater.Open(path);
    this.isNewFile = true;
  }
  else
  {
    this.zipAppender.Open(path);
    this.zipFinder.Open(this.zipAppender);
    this.zipSanner.Open(this.zipAppender);
  }
  this.Password = "[redacted]";
  this.OwnTempFolder = RepoUtil.GetUniqueFolderPath(Path.Combine(tempRootFolder, this.GetHashCode().
  FileSystem.CreateDirectoryIfNotExist(this.OwnTempFolder);
}
```

Firmware format



- Reverse dll for pwd
- Uncompress

- Reverse another dll for pwd
- Uncompress

- Parse and extract .s19 (SREC)
- Identify region types
- Extract/unpack

Confirming assumptions

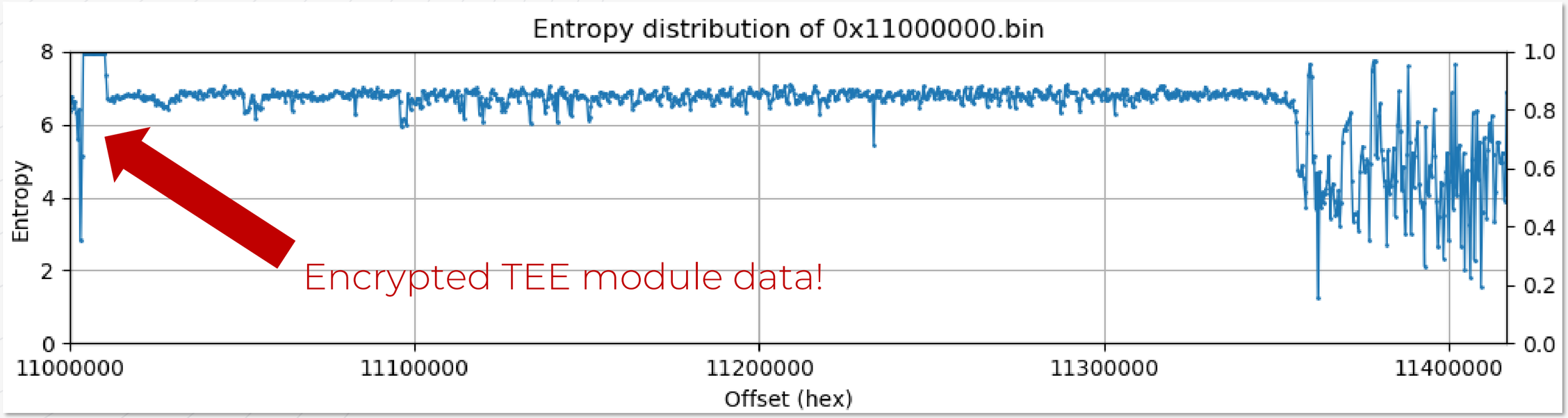
Trusted Execution Environment (TEE) used for TETRA crypto?

- No proper DSP reversing tooling
- How to find out?

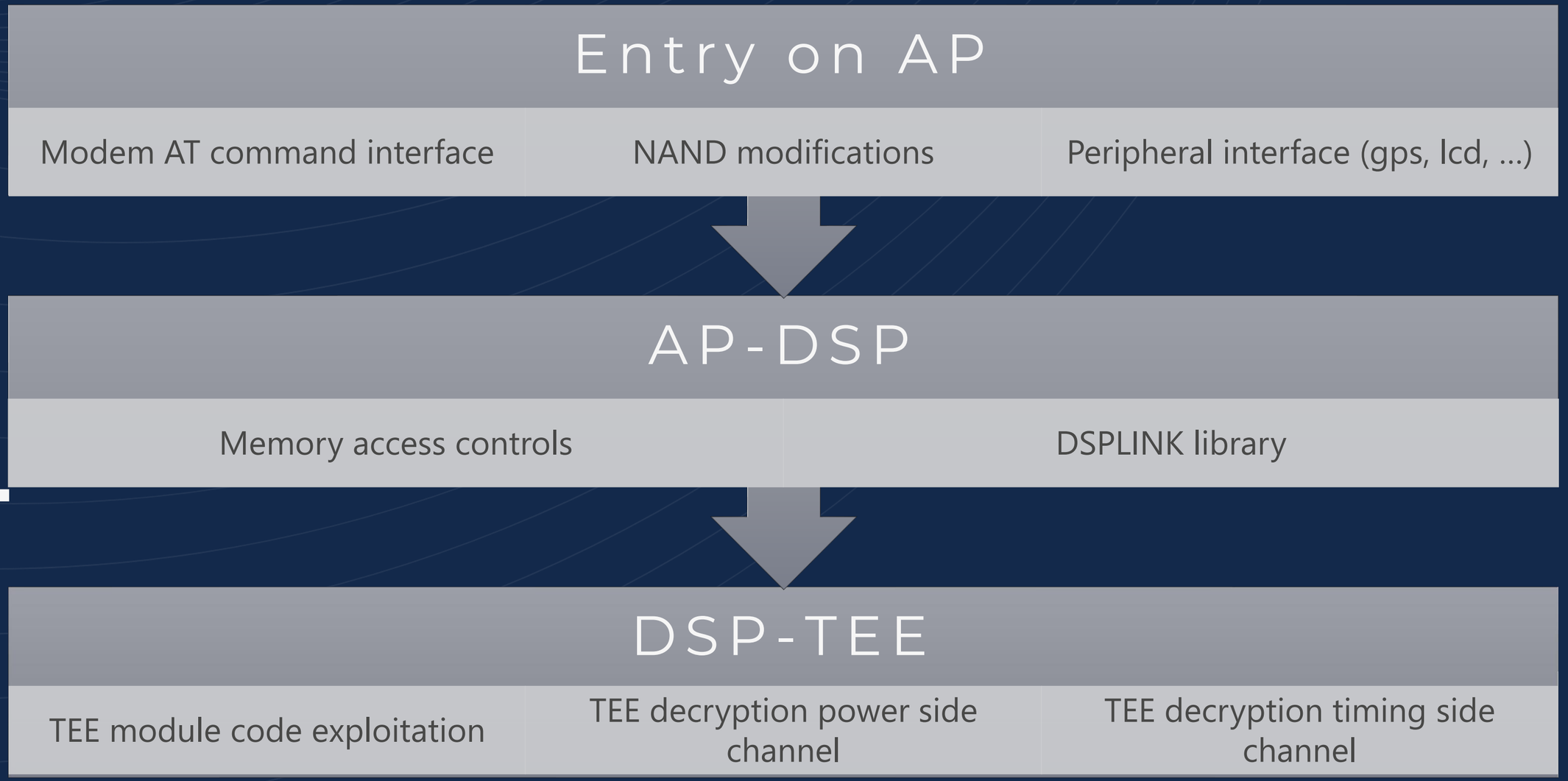
Confirming assumptions

Trusted Execution Environment (TEE) used for TETRA crypto?

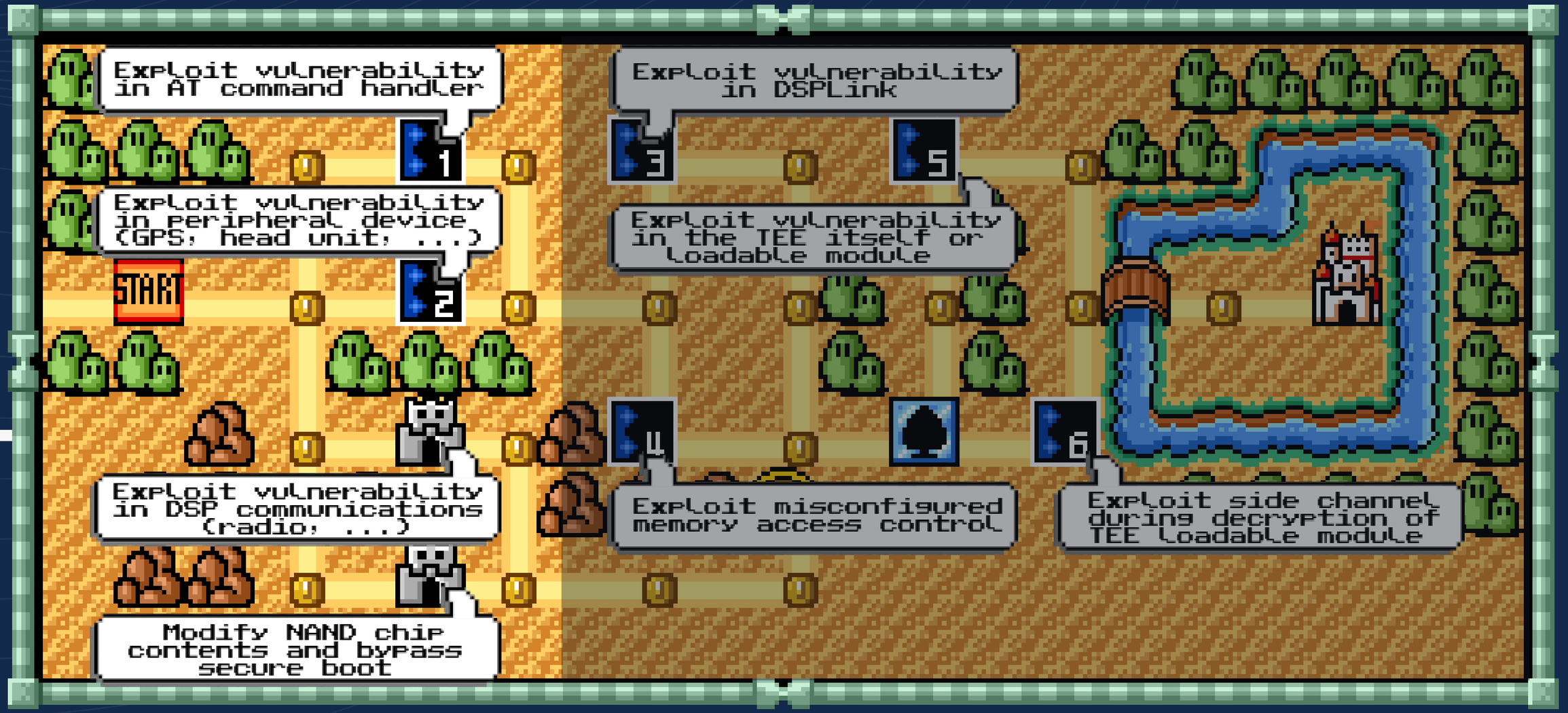
- High-entropy DSP FW area
- TEE API syscall xrefs



The plan


Entry on AP
Modem AT command interface
NAND modifications
Peripheral interface (gps, lcd, ...)
AP-DSP
Memory access controls
DSPLINK library
DSP-TEE
TEE module code exploitation
TEE decryption power side channel
TEE decryption timing side channel

The plan


Exploit vulnerability in AT command handler
1
Exploit vulnerability in peripheral device (GPS, head unit, ...)
START
2
Exploit vulnerability in DSP communications (radio, ...)
Modify NAND chip contents and bypass secure boot
Exploit vulnerability in DSPLink
3
5
Exploit vulnerability in the TEE itself or loadable module
4
6
Exploit misconfigured memory access control
Exploit side channel during decryption of TEE loadable module

AT modem command interface

- Read/write parameters
- Usage commands
- Firmware analysis yields command list
- Want those with strings / variable length handling

```
AT+CNUM      (0-7),(0-10231638316777215),[(Alpha)]
AT+MCDNCN    (1-42 chars),(1-14 chars)
AT+MCDNTN    (1-42 chars),(1-14 chars),(1-300),(0-8)
AT+MCTGDW    (1-2000),(0-12 chars),(0-15999999,16777215),
             (380000000-430000000),(0-999,1023),(0-16383)
```

Format string vulnerability

AT+MCTGDW : Set an entry in the talk group list

AT+CTGL : Enumerate the talk group list

- Classic format string vuln!
  - We can write arbitrary data to stack
- If we control any value on the stack, we can write anywhere
  - Unfortunately, we don't :/

```
if ( bOutputType == 1 )
{
  GetDmoEntry(&szCtlglLine, *(void **)(a1 + 0xC4), i, 1, 0);
  if ( std::string::length(&szCtlglLine) )
  {
    std::string::appendFormat((std::string *)(a1 + 0xEC), "\r\n");
    std::string::appendFormat((std::string *)(a1 + 0xEC), "%d,", bFolderIndex);
    szCtlglLineCharPtr = std::string::c_str(&szCtlglLine);
    std::string::appendFormat((std::string *)(a1 + 0xEC), szCtlglLineCharPtr);// vuln hits
  }
  p_szCtlglLine = &szCtlglLine;
}
```

```
AT+MCTGDW=5,"DMO6%p",2005,390000000,0,0

OK
AT+CtGL?

+CTGL: 1,DMO Folder
1,2000,DMO1
1,2001,DMO2
1,2002,DMO3
1,0000000000002003,DMO4
1,0000000000002005,DMO60x80000000

OK
```

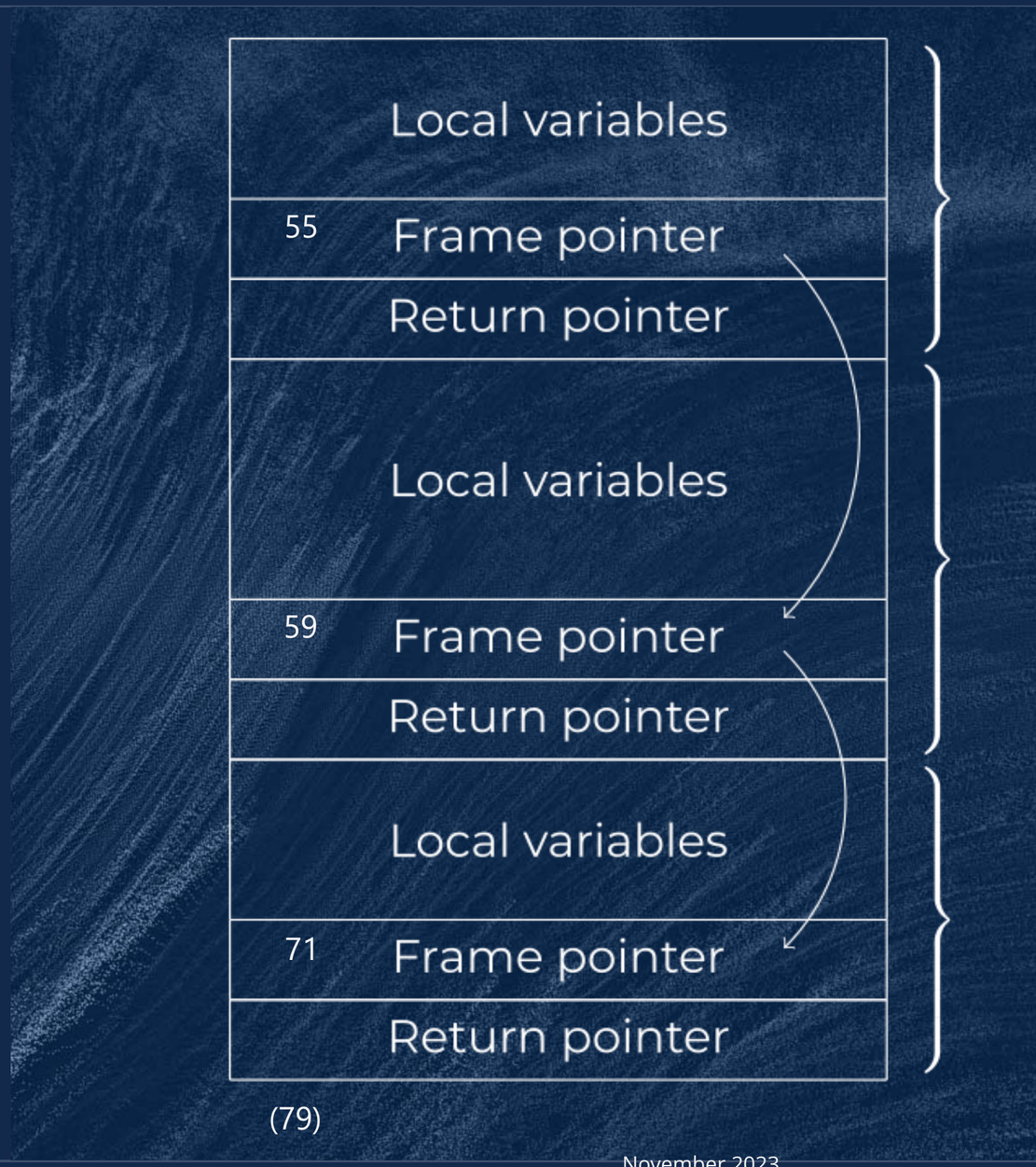
The exploit

- Frame pointers to the rescue!

- Suppose that arg 55 is a frame pointer
  - Part of stack frame chain (linked list)
  - Next FP is arg 59

- Set LSB of arg 59 to 0x80 ("%128c%55$hhn")
  - Now points to byte 0 of arg 79
  - Write ‘A’ to byte 0 ("%65c%59$hhn“)

- Iterate LSB 0..3 to fully control arg 79
  - **Write-what-where** (“%66c%79$hhn”)



Making it work

- Now have write-what-where
- Turns out that the heap is RWX
  - should be easy
- Approach:
  - Write shell code to heap
  - Overwrite some pointer to executable code
  - Trigger execution of that pointer
  - Profit

[+] Got a root shell on the Application Processor

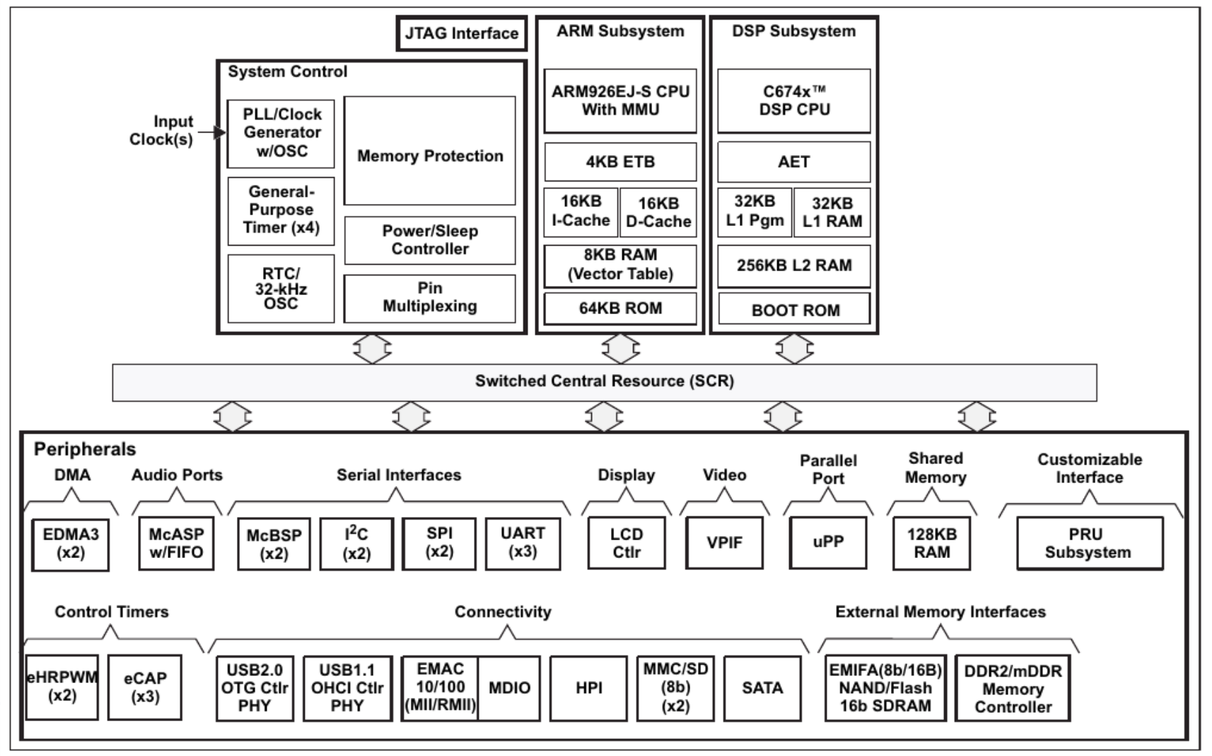
```
route -n
/tmp # Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0.2
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
dmesg | tail
/tmp # [    2.730000] yaffs: passed flags ""
[    2.730000] yaffs: Attempting MTD mount on 31.2, "mtdblock2"
[    2.730000] yaffs: restored from checkpoint
[    2.730000] yaffs_read_super: isCheckpointed 1
[    2.860000] Serial: 8250/16550 driver, 3 ports, IRQ sharing disabled
[    2.860000] serial8250.0: ttyS0 at MMIO 0x1c42000 (irq = 25) is a 16550A
[    2.860000] serial8250.0: ttyS1 at MMIO 0x1d0c000 (irq = 53) is a 16550A
[    2.860000] serial8250.0: ttyS2 at MMIO 0x1d0d000 (irq = 61) is a 16550A
[    2.970000] ToolDev : application firestarter found, pid: 16
[    3.370000] eth0: attached PHY driver [Generic PHY] (mii_bus:phy_addr=1:00, id=
df -h
/tmp # Filesystem                Size      Used Available Use% Mounted on
/dev/root                10.8M     10.8M         0 100% /
tmpfs                   128.0K      4.0K    124.0K   3% /var/run
tmpfs                    16.0M      2.0M     14.0M  13% /tmp
/dev/ucfg                 2.5M      1.3M      1.2M  50% /mnt/ucfg
/dev/codeplug            41.8M     15.1M     26.7M  36% /mnt/flash
/dev/keys                 2.0M      1.1M    876.0K  57% /mnt/keys
cat /proc/cmdline
/tmp # mem=32M console=/dev/null root=/dev/ram0 initrd=0xc1000000,5M ramdisk_size=5
```

- Root shell achieved

The plan


Exploit vulnerability in AT command handler
Exploit vulnerability in peripheral device (GPS, head unit, ...)
START
Exploit vulnerability in DSP communications (radio, ...)
Modify NAND chip contents and bypass secure boot
Exploit vulnerability in DSPLink
Exploit vulnerability in the TEE itself or loadable module
Exploit misconfigured memory access control
Exploit side channel during decryption of TEE loadable module

A tale of two cores



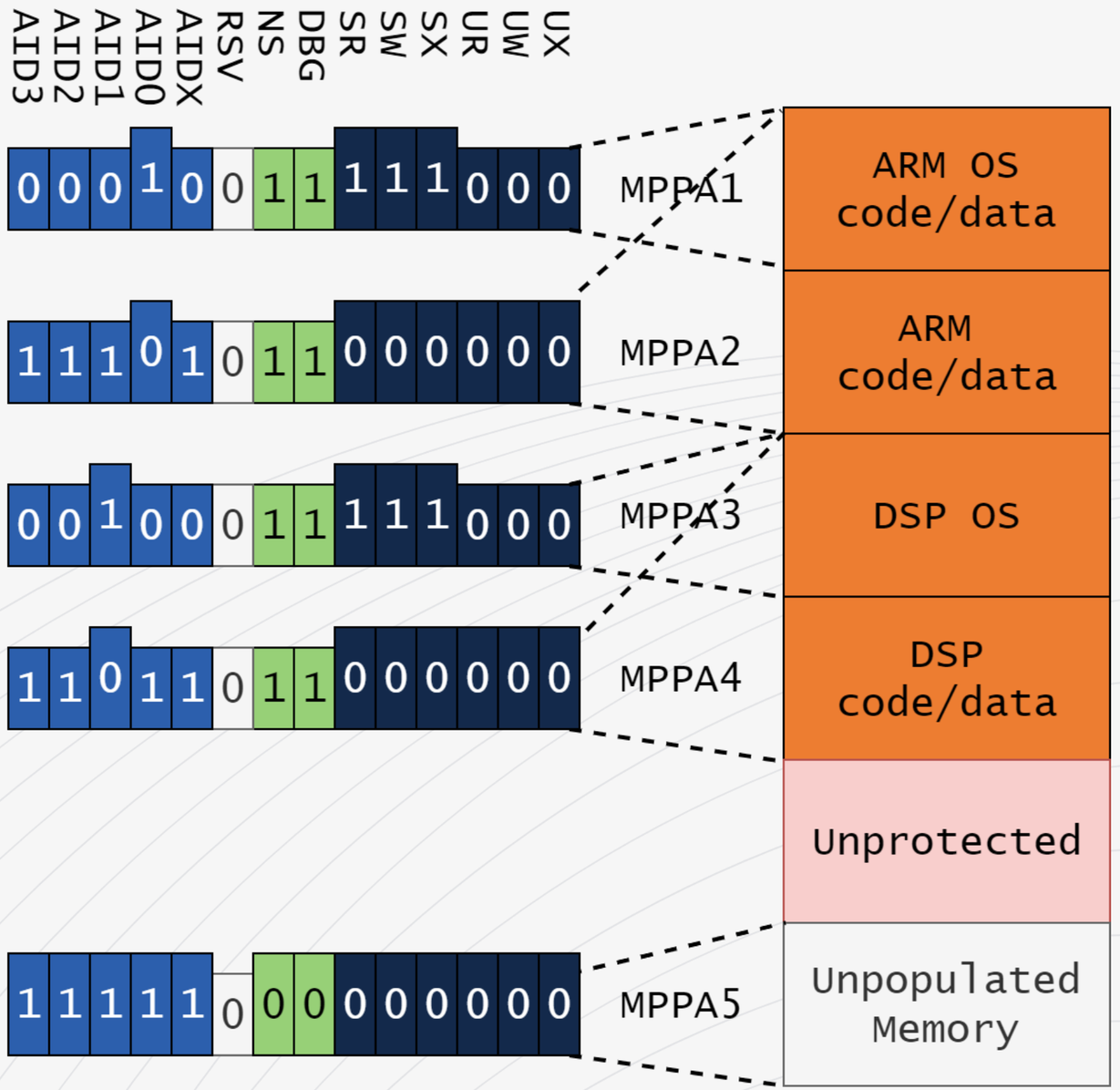
Source: OMAP-L138 C6000 DSP+ARM Processor datasheet (SPRS586J)

- AP: ARM
- DSP: C674x
- Shared:
- 128KB internal RAM
- External DDR2 memory

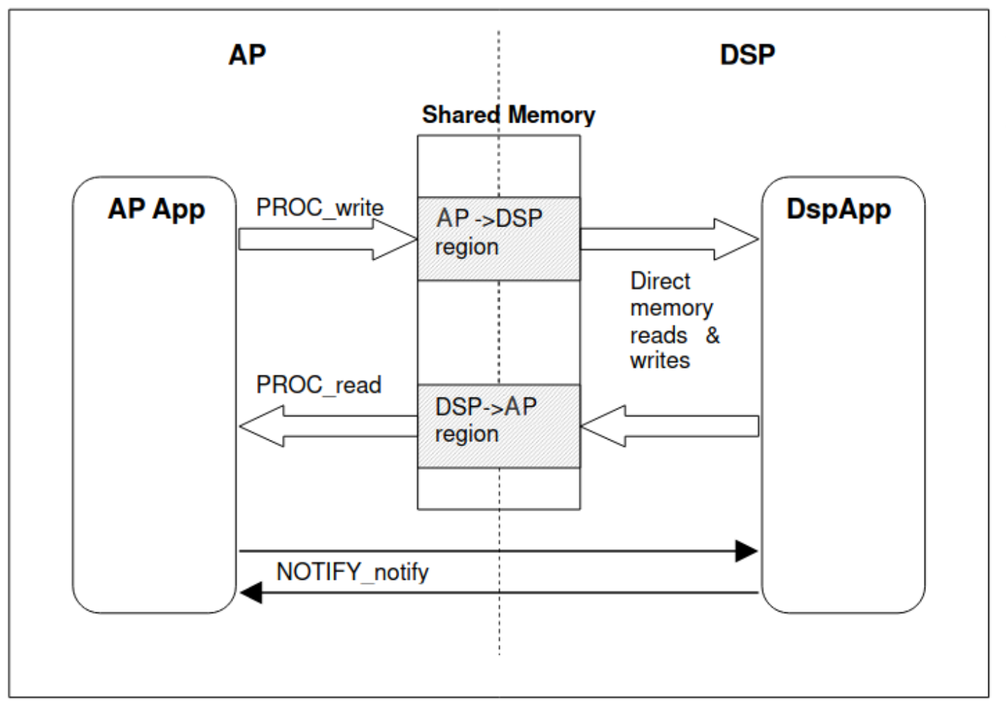
- We somehow need to move laterally and gain control over the DSP.
  - Abuse shared memory/race condition?

Memory protection

- Memory must be segmented to prevent access by random unauthorized cores/devices
- MPUs implement a ‘firewall’
- OMAP-L138 has MPUs



The plan



Source: DSP/BIOS™ Link user guide

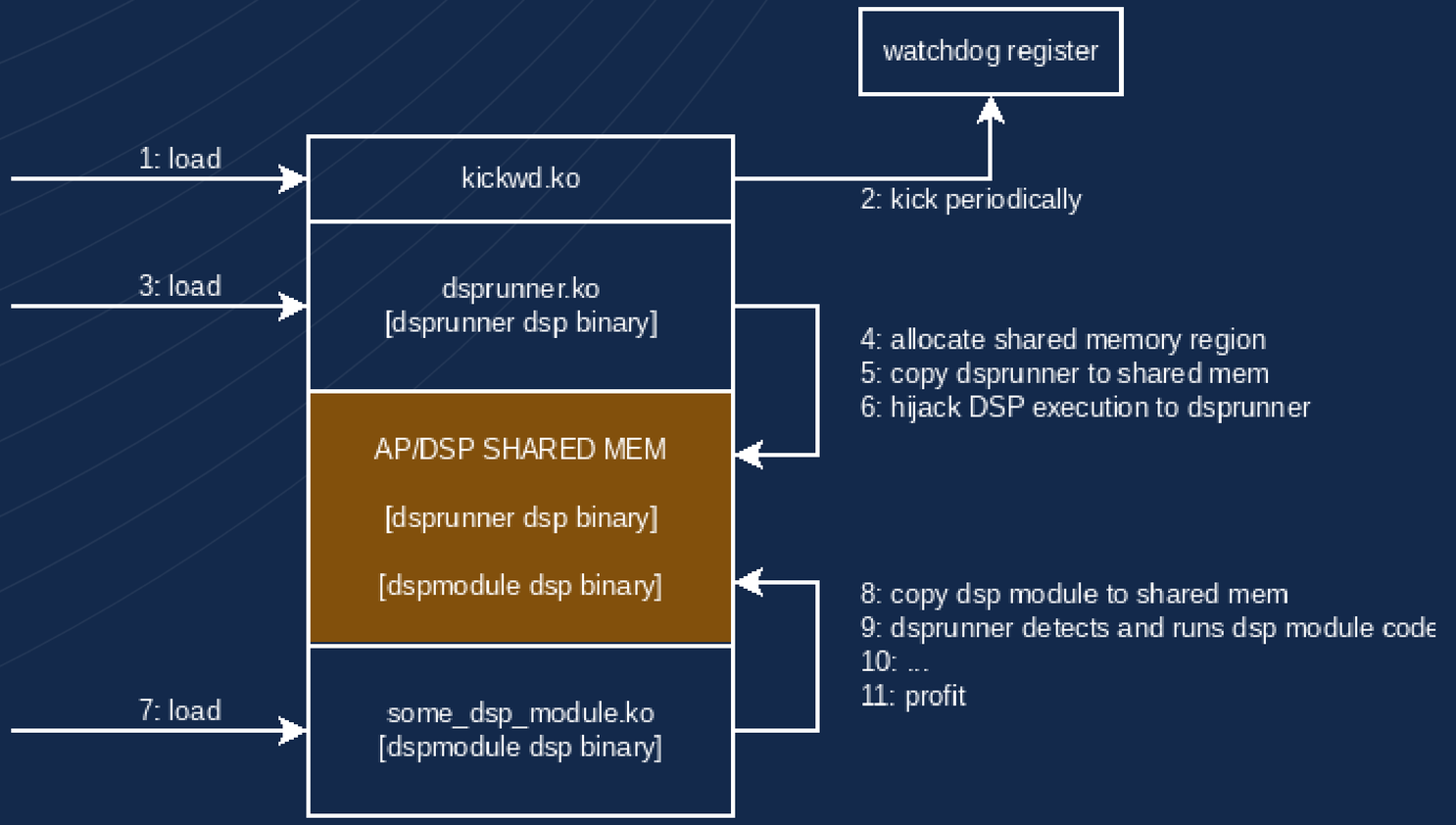
- Dump MPU config
- Find ranges that are:
  - Used by DSP
  - Writeable by AP
- Identify and exploit an interface that uses such a shared memory range
  - Example: DSPLINK

```
[i] Dump MPU1 config (addr 01E14000 len 8)
[+] Got rev 0149814E conf 01C00600
    00000000 00000110 11000000 00000001
    |------|                                address alignment 2^0KB == 1KB
           |--|                               num fixed ranges: 0
              |--|                            num programmed ranges: 6
                                 |            default: ASSUME_ALLOWED

[i] Dump MPU1 range 1 (addr 01E14200 len 12)
[+] Got 00000000 FFFFFFFF FFFEFF03
       |------|                               range start: addr 0
              |------|                        range end: addr ffffffff
                     |------|                 permission flags
    Permissions:
    00000011111111111111011111111
       |-------------|                        applies to AP and DSP
                     |----|                   all permissions
```

Taking control of the DSP

- Implemented framework for running code on DSP
- Multiple AP kernel modules
- Hijack control of DSP
- Prevent board reset
- Set up shared physical memory region



MIDNIGHT BLUE

DSP reversing

- Hellish architecture
  - Delayed branches
  - Variable degree of concurrency
  - Conditionals
- No IDA support
- DIY time!
  - Implemented disassembler
  - Ultimately, also a decompiler


```
.DSP_L2_ROM:007F8B20
.DSP_L2_ROM:007F8B20 SK_LOAD_handler_Impl:                    ; CODE XREF: SK_LOAD_handler+A↓j
.DSP_L2_ROM:007F8B20                    MVC     .S2     NTSR, B0
.DSP_L2_ROM:007F8B24                    MVC     .S2     TSR, B1
.DSP_L2_ROM:007F8B28                    EXTU    .S2     B0, 31, 30, B0
.DSP_L2_ROM:007F8B2C                    OR      .D2     B0, B1, B1
.DSP_L2_ROM:007F8B30                    MVC     .S2     B1, TSR
.DSP_L2_ROM:007F8B34                    RINT
.DSP_L2_ROM:007F8B38
.DSP_L2_ROM:007F8B38                    MVK     .S1     (g_dwDelegateKeyNo & 0FFFFh), A0
.DSP_L2_ROM:007F8B3C            ||      STW     .D2     A13, *B15--[10h]
.DSP_L2_ROM:007F8B40
.DSP_L2_ROM:007F8B40                    MVKH    .S1     unk_800000, A0
.DSP_L2_ROM:007F8B44            ||      STW     .D2     A14, *+B15[0Dh]
.DSP_L2_ROM:007F8B48
.DSP_L2_ROM:007F8B48                    LDBU    .D1     *+A0, A0
.DSP_L2_ROM:007F8B4C            ||      STW     .D2     B3, *+B15[0Ch]
.DSP_L2_ROM:007F8B50
.DSP_L2_ROM:007F8B50                    STW     .D2     B13, *+B15[0Bh]
.DSP_L2_ROM:007F8B54                    STW     .D2     A12, *+B15[0Ah]
.DSP_L2_ROM:007F8B58                    STW     .D2     B12, *+B15[9]
.DSP_L2_ROM:007F8B5C                    STW     .D2     A11, *+B15[8]
.DSP_L2_ROM:007F8B60
.DSP_L2_ROM:007F8B60              [A0]  B       .S2     loc_7F9004
.DSP_L2_ROM:007F8B64            ||[A0]  MVK     .L1     4, A0
.DSP_L2_ROM:007F8B68
.DSP_L2_ROM:007F8B68                    STW     .D2     B11, *+B15[7]
.DSP_L2_ROM:007F8B6C                    STW     .D2     A10, *+B15[6]
.DSP_L2_ROM:007F8B70                    STW     .D2     B10, *+B15[5]
.DSP_L2_ROM:007F8B74                    STW     .D2     A15, *+B15[4]
.DSP_L2_ROM:007F8B78
.DSP_L2_ROM:007F8B78                    STW     .D2     B4, *+B15[3]
.DSP_L2_ROM:007F8B7C            ||      MV      .D1X    B15, A15
.DSP_L2_ROM:007F8B7C                    ; BRANCH loc_7F9004 OCCURS
.DSP_L2_ROM:007F8B80
.DSP_L2_ROM:007F8B80                    SUBAW   .D2     B15, 0Ah, B15
.DSP_L2_ROM:007F8B84                    ADDAW   .D1     B15, 2, A11
.DSP_L2_ROM:007F8B88
.DSP_L2_ROM:007F8B88                    MV      .D2X    A11, B13
.DSP_L2_ROM:007F8B8C            ||      MVK     .S2     2Ch, B0
.DSP_L2_ROM:007F8B90
.DSP_L2_ROM:007F8B90                    SUBAW   .D2     B15, B0, B15
.DSP_L2_ROM:007F8B94                    ADDAW   .D1     B15, 2, A6 ; lpDecryptCtx
.DSP_L2_ROM:007F8B98                    STW     .D1     A6, *+A15[0Eh]
.DSP_L2_ROM:007F8B9C                    MV      .D1     A4, A13
.DSP_L2_ROM:007F8BA0
.DSP_L2_ROM:007F8BA0                    MVK     .S1     (g_abCek & 0FFFFh), A4
.DSP_L2_ROM:007F8BA4            ||      SUBAW   .D2     B15, B0, B15
.DSP_L2_ROM:007F8BA8                                    ; lpKey
```

```
30 // Address range: 0x7f8b20 - 0x7f903c
31 int32_t SK_LOAD_handler_Impl(int32_t result, int32_t a2, int32_t a3, int32_t a4, ir
32     // 0x7f8b20
33     __SET_INTERRUPTS_ENABLED(true);
34     if (*(char *)&g_dwDelegateKeyNo != 0) {
35         // 0x7f9004
36         __SET_INTERRUPTS_ENABLED(false);
37         return result;
38     }
39     // 0x7f8b80
40     int32_t v1; // bp-104, 0x7f8b20
41     int32_t v2 = &v1; // 0x7f8b80
42     char * v3 = (char *)(v2 - 168); // 0x7f8b98
43     AES_keyschedule(g_abCek, (char *)(v2 - 344), v3);
44     int32_t v4; // bp-96, 0x7f8b20
45     int32_t v5 = &v4; // 0x7f8bbc
46     int32_t v6 = result; // 0x7f8bbc
47     for (uint32_t i = 6; i >= 0xffffffff; i--) {
48         int32_t v7 = v6;
49         v6 = v7 + 4;
50         *(int32_t *)v5 = *(int32_t *)v7;
51         v5 += 4;
52     }
53     // 0x7f8bd0
54     AES_decrypt((char *)&v4, (char *)&v4, v3);
55     int32_t v8; // bp-80, 0x7f8b20
56     int32_t result2 = AES_decrypt((char *)&v8, (char *)&v8, v3); // 0x7f8bec
```

- Reliable DSP code execution!

Mission accomplished

The plan


Exploit vulnerability in AT command handler
Exploit vulnerability in DSPLink
Exploit vulnerability in peripheral device (GPS, head unit, ...)
START
Exploit vulnerability in the TEE itself or loadable module
Exploit vulnerability in DSP communications (radio, ...)
Exploit misconfigured memory access control
Exploit side channel during decryption of TEE loadable module
Modify NAND chip contents and bypass secure boot

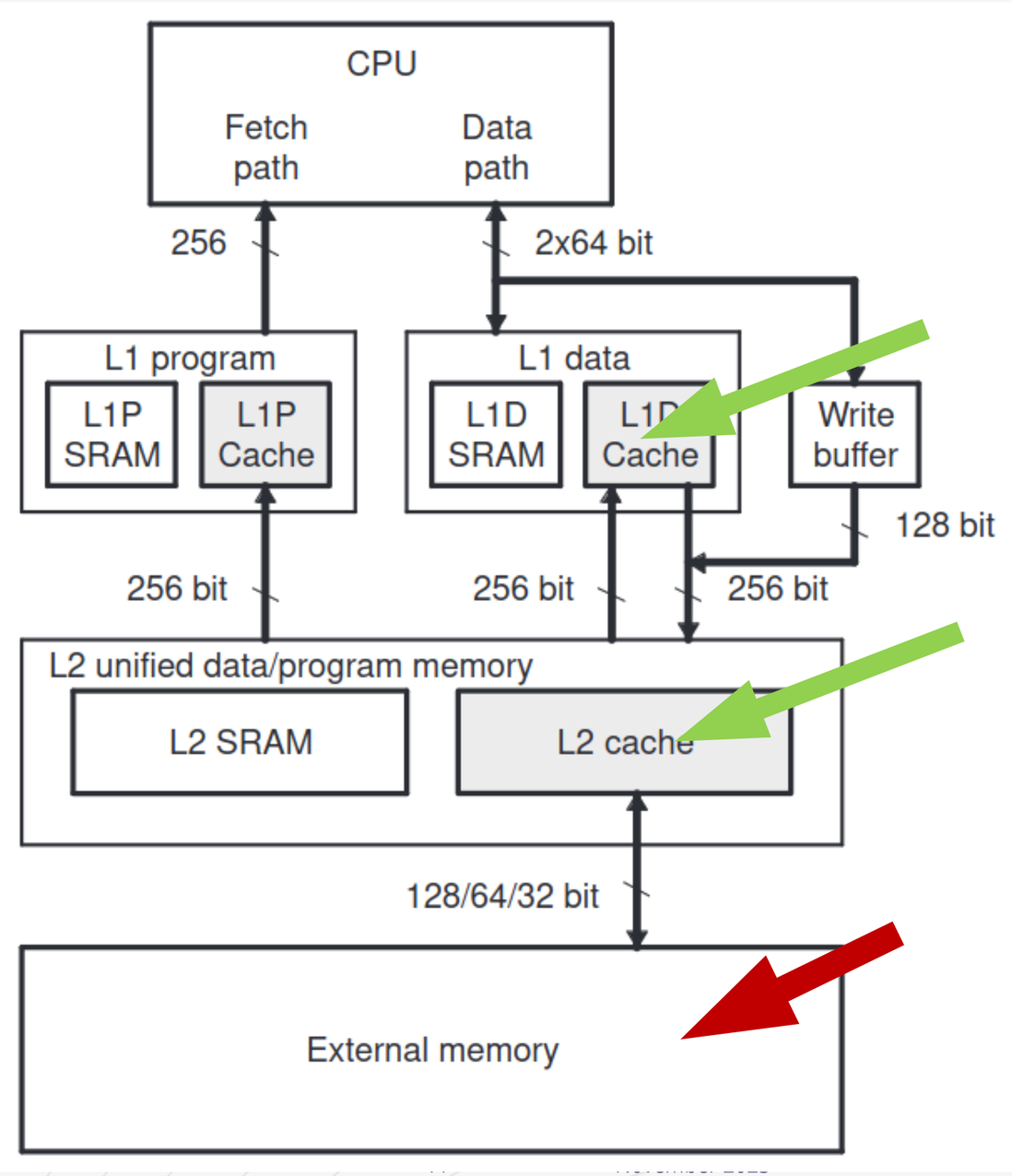
DSP Trusted Execution Environment

Privilege / Security	Secure	Non-secure
Supervisor	Secure Kernel and Secure Boot Loader	DSP/BIOS or other OS kernels
User	Licensed Algorithms (e.g. WMV, Wma, etc...)	Non-secure Applications or other OS kernels

- **TI ROM code implements Secure Kernel**
  - Runs in secure mode
  - Non-secure code can do Secure Kernel API Calls
  - **`SK_LOAD`** call allows for runtime loading of encrypted (**TETRA**) module. Decrypted (**AES**) and validated (**RSA**), then loaded as TEE module
  - **`SK_ALGOINVOKE`** call allows non-secure code to invoke a function of a loaded TEE module.
  - Module encrypted with factory-set Customer Encryption Key (**CEK**)

DSP cache architecture

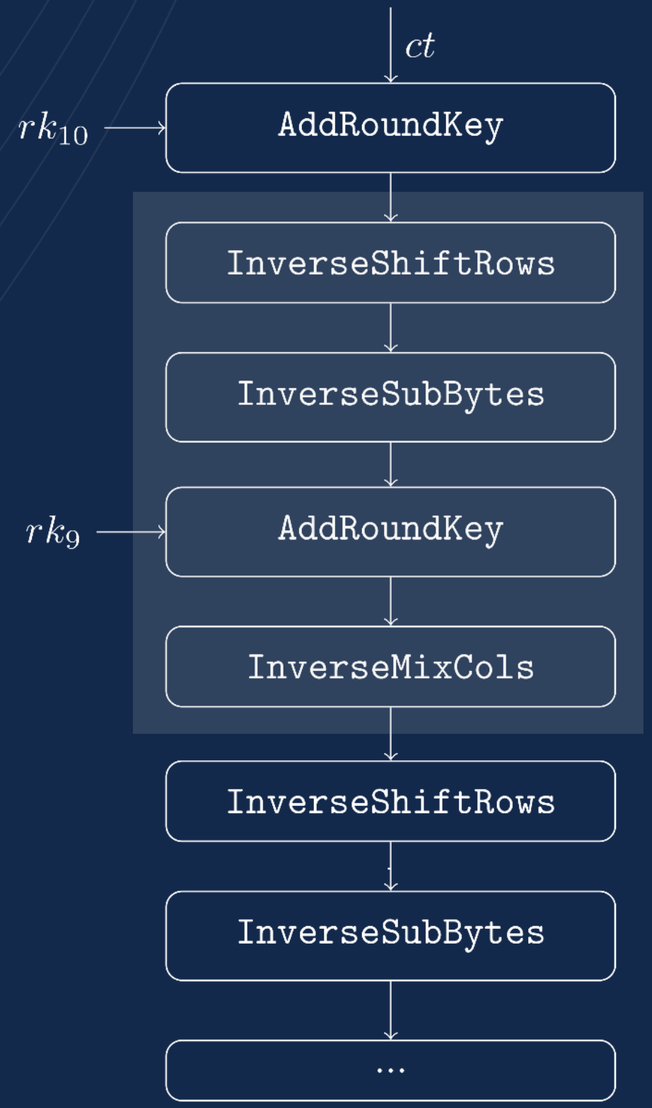
- L1D, L1C, L2 caches
- L1D data cache: 64-byte lines
- Memory read is..
  - **Fast** if already in cache
  - **Slow** if not in cache
- We can:
  - Evict addresses from cache
  - Freeze cache (switch to read-only)

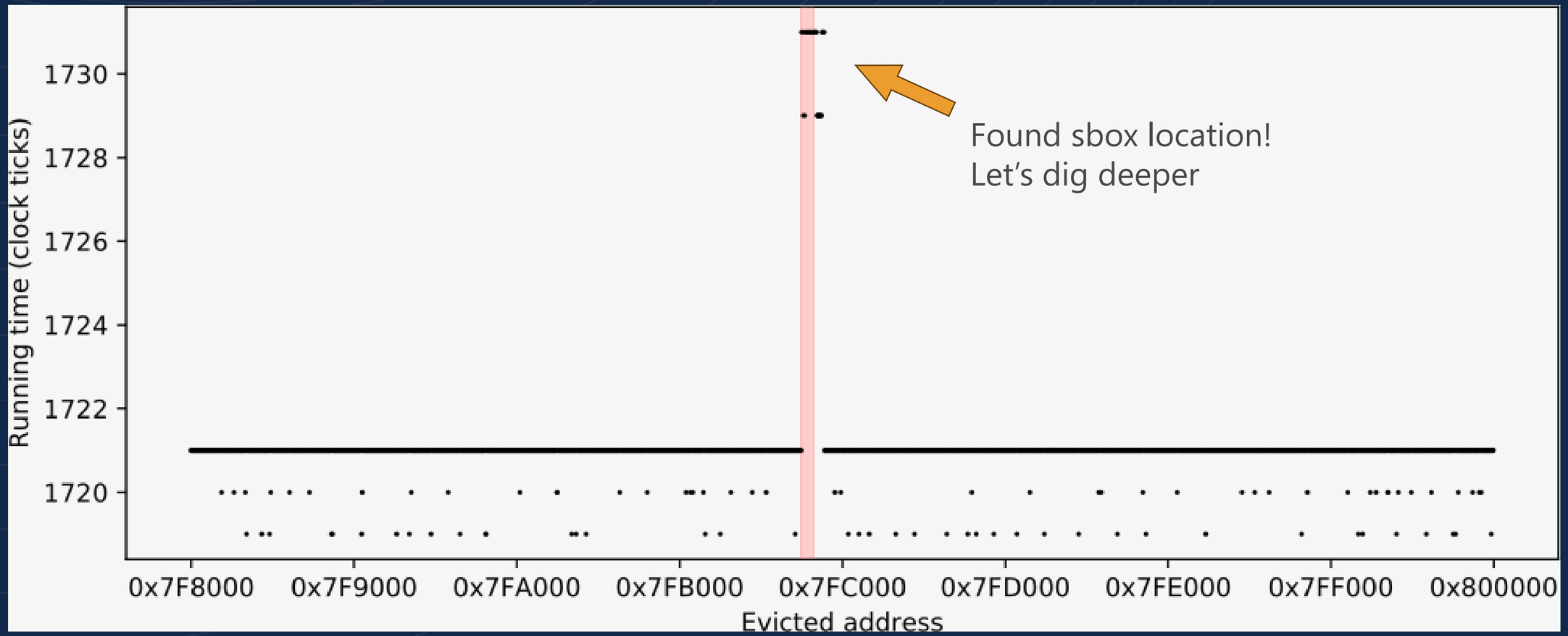


* TMS320C674x DSP Cache User's Guide (SPRUG82A)

Next step?

- We have some control over cache
- Which AES decryption step could we target?
  - InverseSubBytes uses an sbox
  - Memory lookup
  - But where is it?
- Evict secret ROM page line from cache
- Measure `SK_LOAD` running time
  - Repeat for each line
- Plot




Running time (clock ticks)
1730
1728
1726
1724
1722
1720
0x7F8000
0x7F9000
0x7FA000
0x7FB000
0x7FC000
0x7FD000
0x7FE000
0x7FF000
0x800000
Evicted address
Found sbox location!
Let's dig deeper

Sbox cache state

- **Prepare cache as follows:**
  - Run SK_LOAD: full sbox in cache
  - Evict first 32 sbox entries (first octant)
  - Enable cache freeze

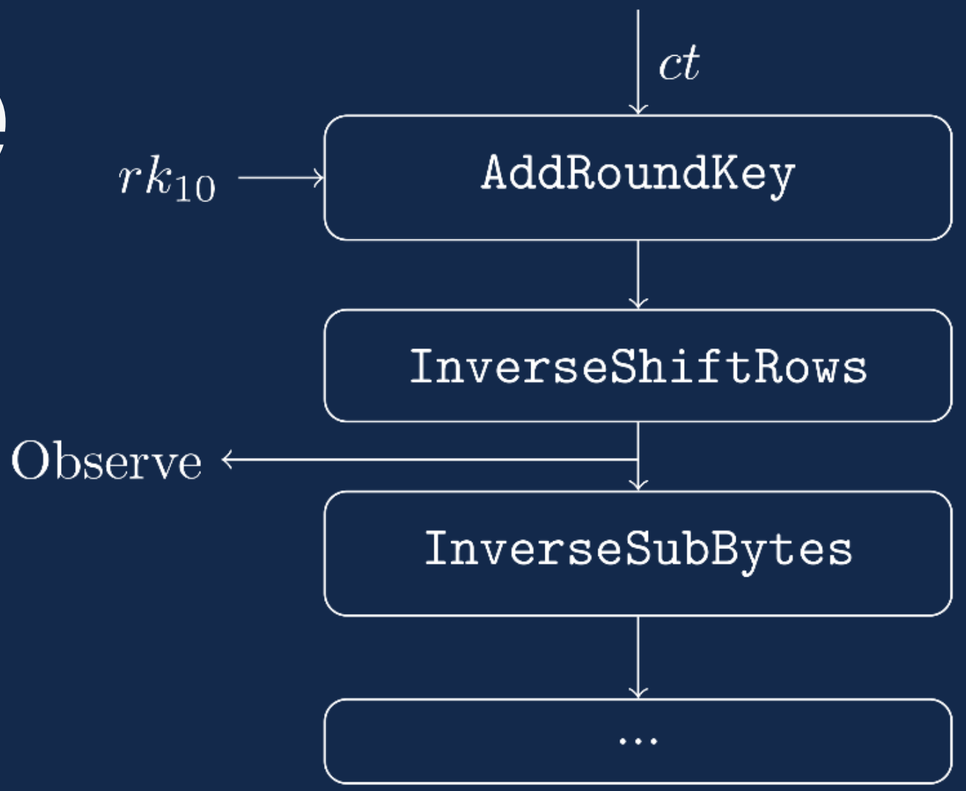
- *We now observe a timing penalty whenever an sbox lookup hits the first sbox octant*

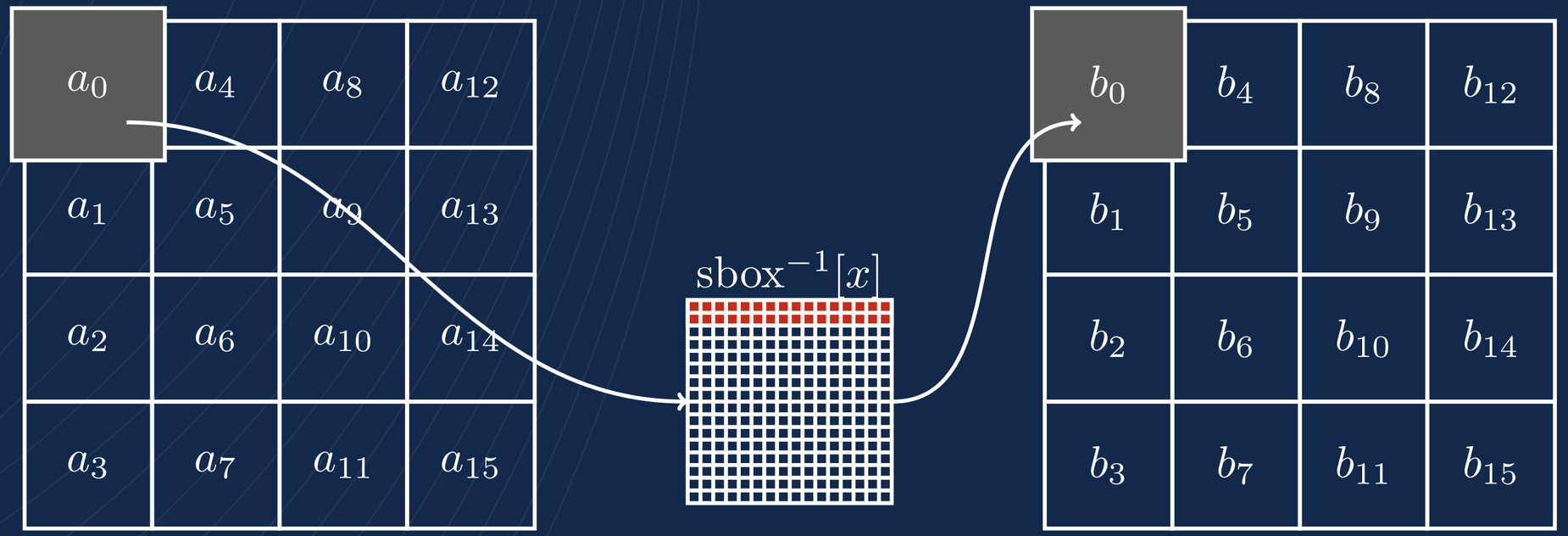
- *This setup is assumed for the remainder of the presentation*

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
10	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
20	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
30	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
40	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
50	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
60	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
70	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
80	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
90	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a0	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18		
b0	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78			
c0	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27		ec	5f
d0	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93		9c	ef
e0	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	8			
f0	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	5			

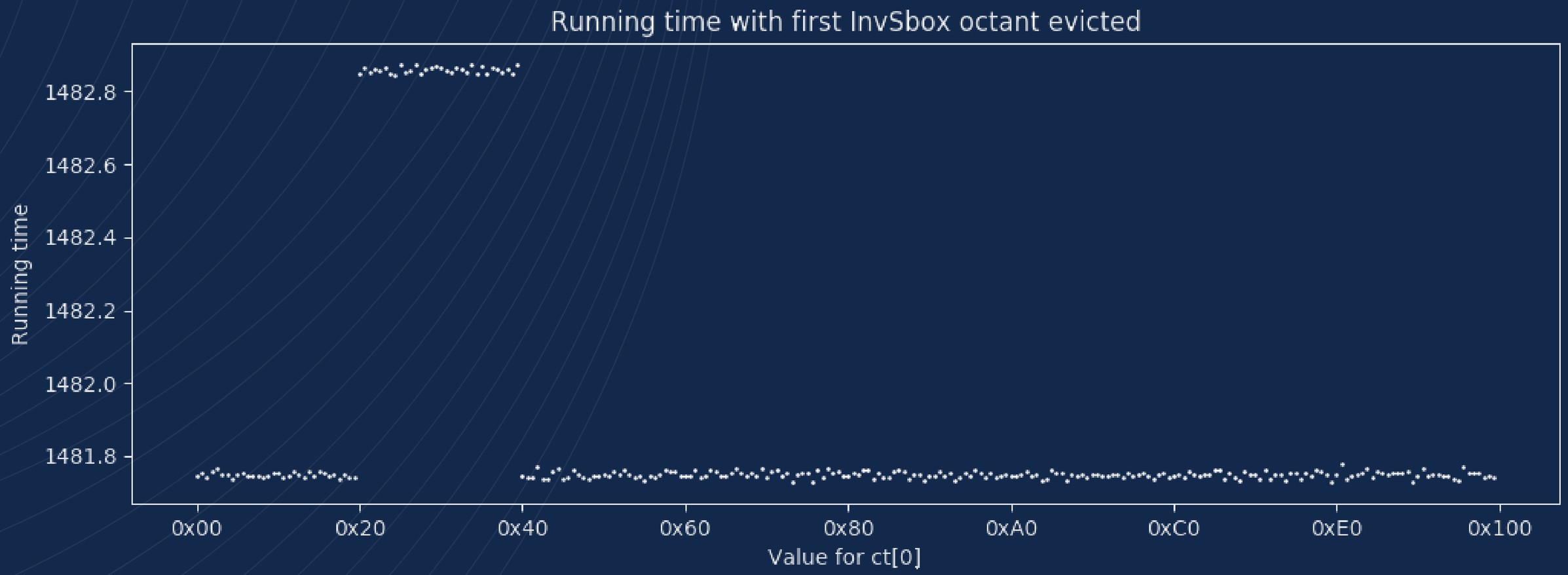
Attack outline

- Having evicted the first octant of sbox from cache (32 entries)...
- Set $ct[0]$ to 0
- Randomize remainder of ct
- Get average running time of `SK_LOAD`
- Repeat for other values of $ct[0]$
- Plot





- Set $ct[0]$ to 0
- Randomize remainder of ct
- Get average running time of `SK_LOAD`



- If penalty observed: $ct[0] \oplus rk_{10}[0] < 0x20$
- Above example: $0x20 \leq rk_{10}[0] < 0x40$

Success!

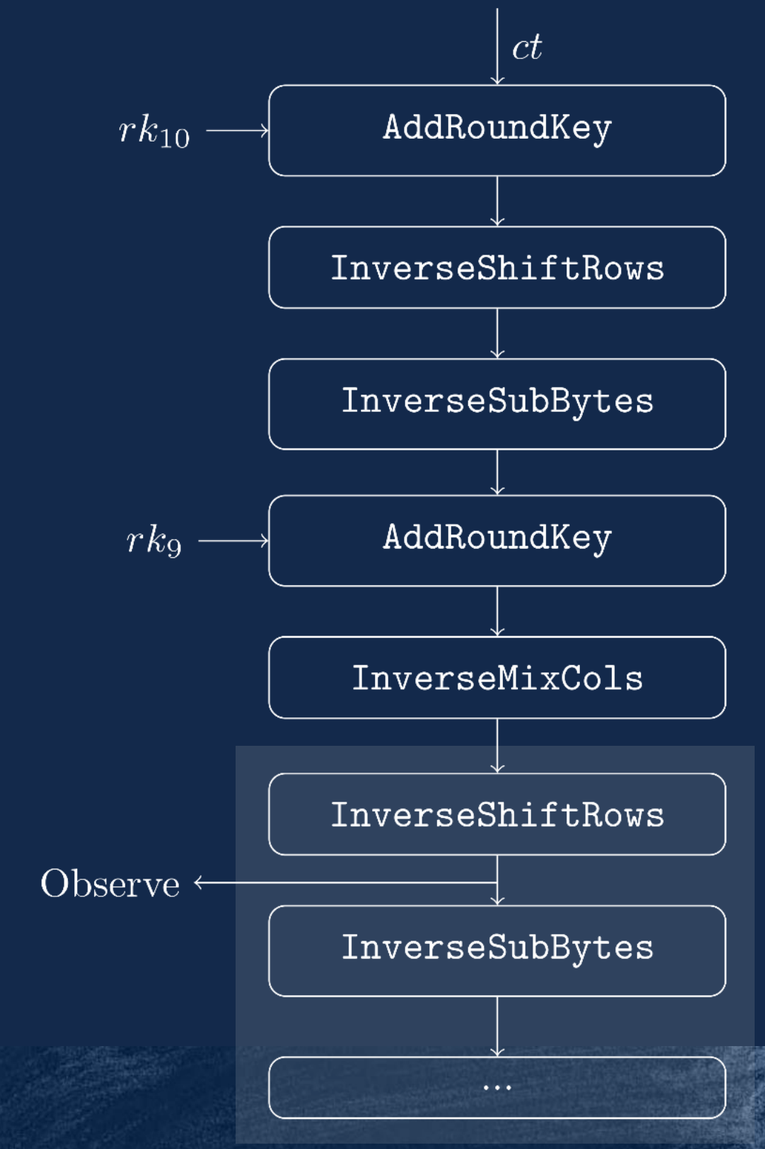
- In our example: 0x20 $\leq rk_{10}[0] <$ 0x40
- Effectively 3 bits of $rk_{10}[0]$
- Can repeat to obtain 48 bits of rk_{10} !

Success..?

- Inherently limited
  - Least significant 5 bits of $rk_{10}[0]$ do not influence which sbox octant is hit
  - Still missing 80 bits
  - Too much for exhaustive search
  - We need to go deeper

Extended attack

- **Take attack one round further**
  - Observe penalties during round 2 `InverseSubBytes`
- **Round 1 `InverseSubBytes` has introduced required diffusion within the state byte**
  - Least significant bits of rk_{10} byte now influence the most significant bits of the state byte
- **Need to account for:**
  - Shifting of state bytes
  - Influence of rk_9
  - Diffusion over four state bytes from `InverseMixCols`



Demo: CVE-2022-25332

SK_LOAD Cache Timing Side-Channel

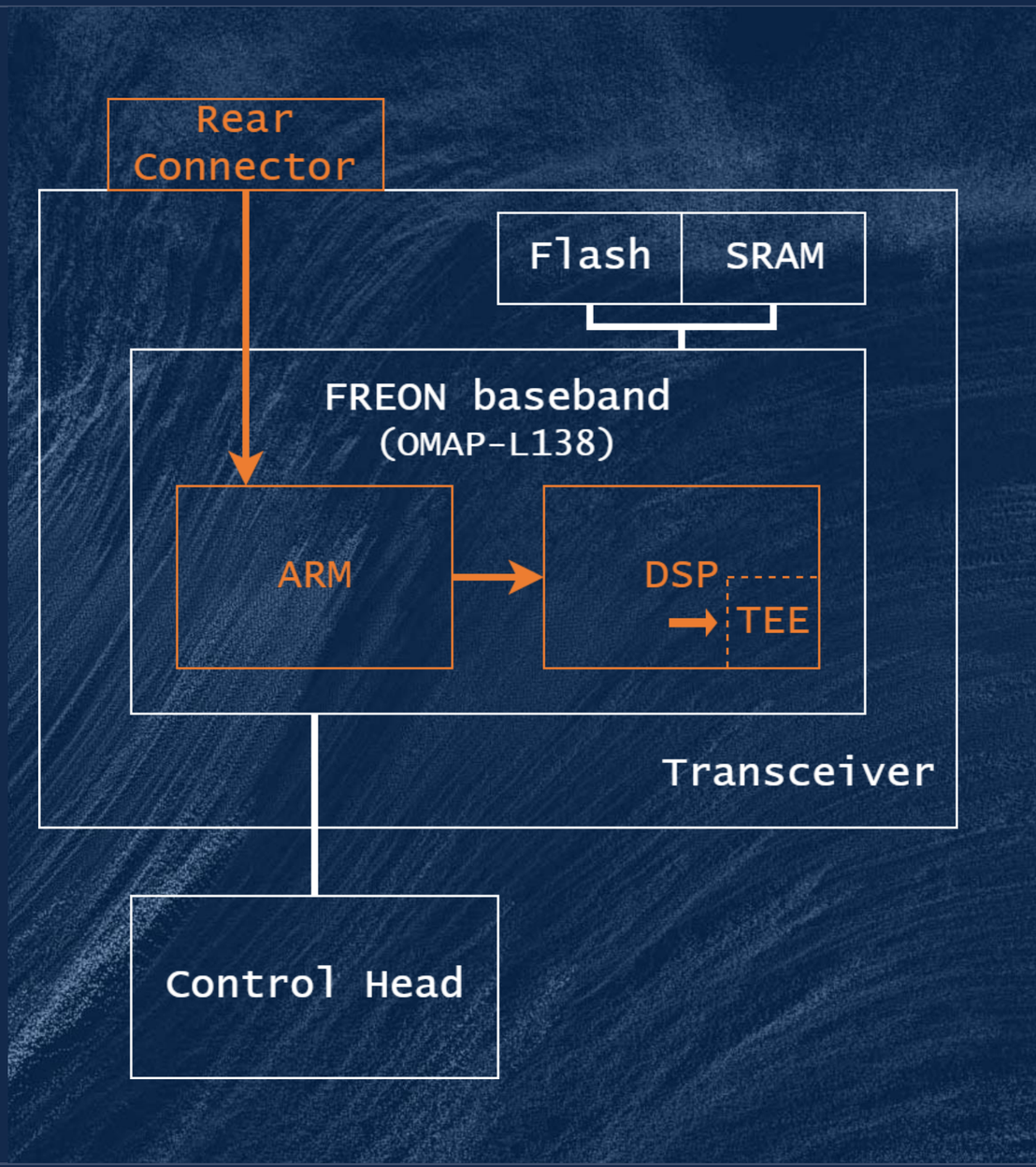
Result

- Can extract CEK in < 1 min
- Reverse-engineered remainder of module format
  - Different section keys
  - Obfuscation scheme
- Decrypted SKLOAD module!

- Reverse-engineered SKLOAD module
- Generated TETRA crypto test vectors on LCDK
- Distilled specification, implemented C reference code

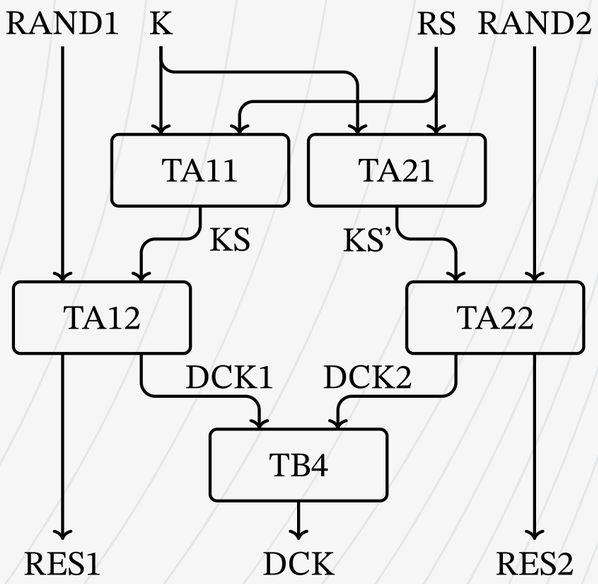
Recap

1. Format string → code exec on AP
2. Pivot to DSP via shared memory
3. Cache timing side-channel on TEE
4. …
5. Secret algos!
   … and key extraction …



The secret TETRA primitives and their security

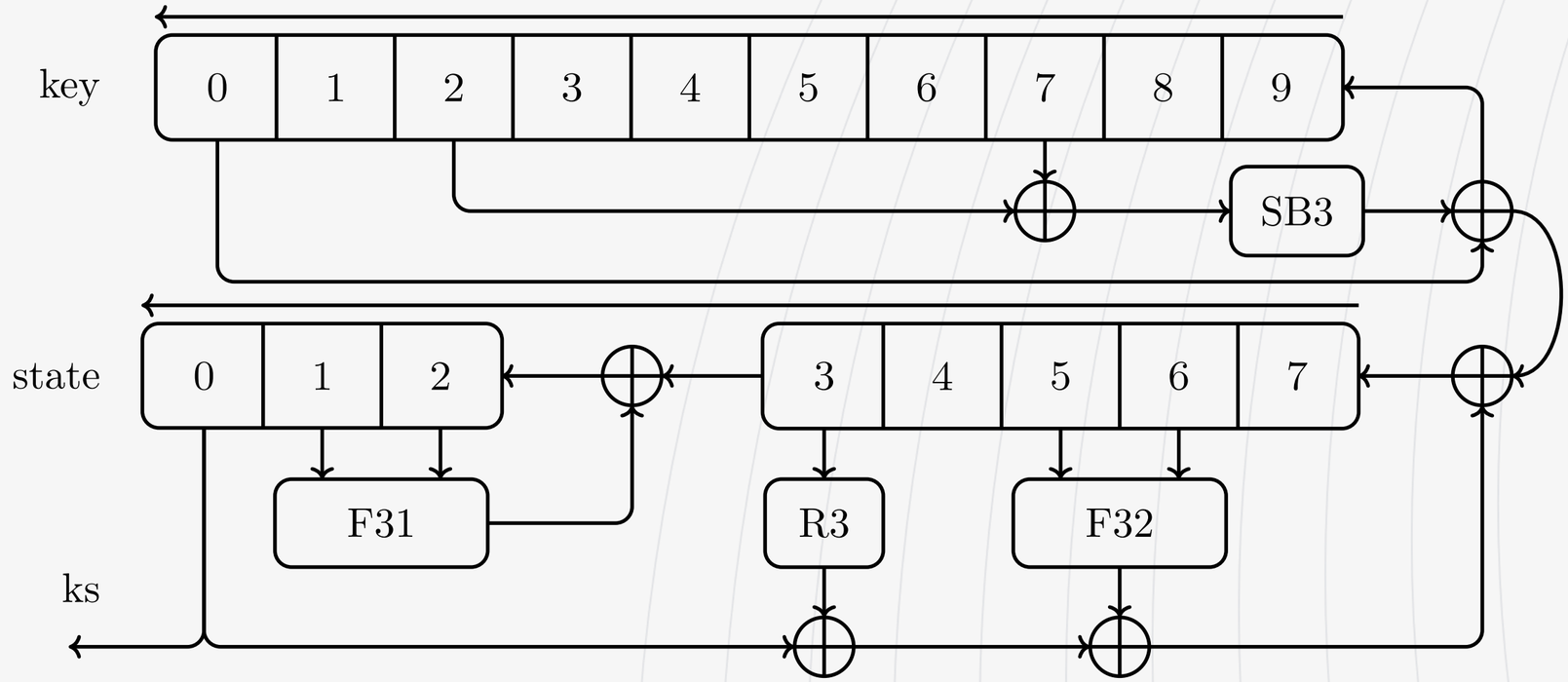
TA1 auth and OTAR



- Protocols in public standard, primitives not.
- Some blocks turn out identical or related
  - TA11 = TA41
  - TA12 = TA22
  - TA11(K, RS) = TA21(K, reversed(RS))
- Found two issues
  - DCK pinning attack (CVE-2022-24400)
  - De-anonymization (CVE-2022-24403)

TEA Keystream generators

- Used for air interface encryption
- All KSGs have similar structure
- TEA2 seems robust*
  - We are not cryptographers
  - Public scrutiny needed!



Pictured: TEA2

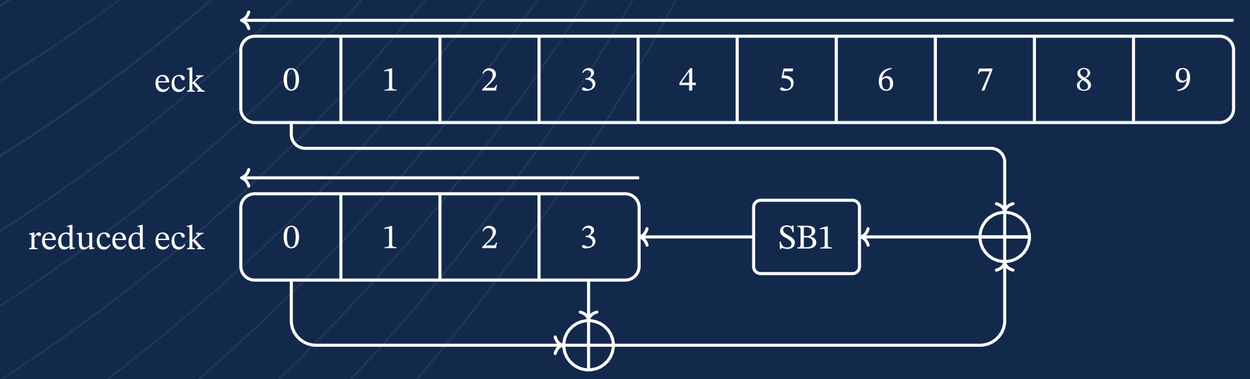
CVE-2022-24402 TEA1 backdoor

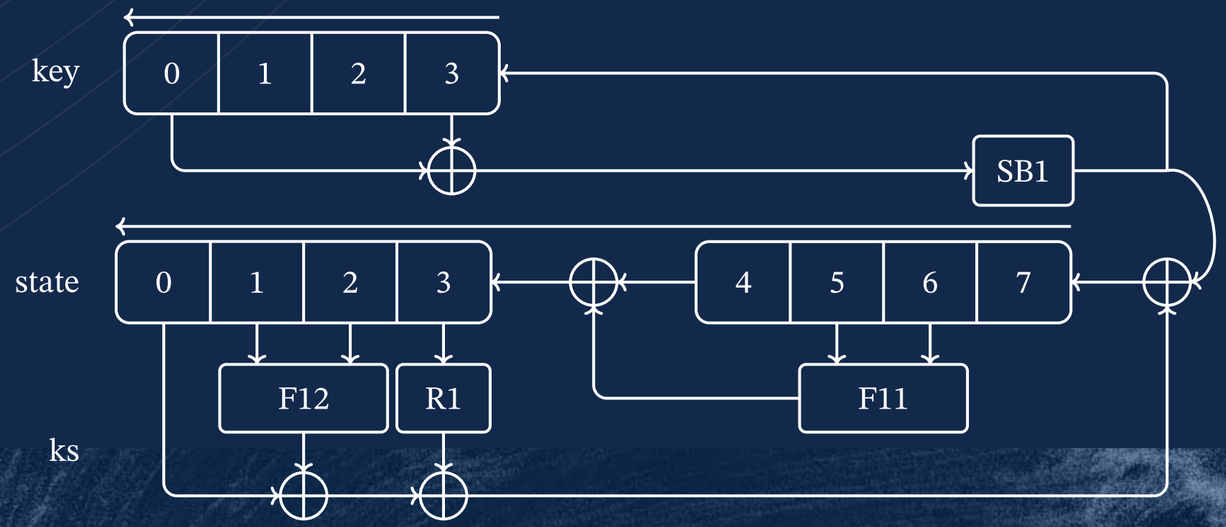
- **Target audience**
  - Private security, “less friendly” LE / mil
  - .. But also, power, water, oil & gas

- **Advertised with 80-bit key**
  - Readily exportable but no hard indication on actual security (56-bit? 40-bit?)

- **Has “key initialization” function**
  - Reduces 80-bit key into 32-bit register

- **Trivial passive brute force (<1min)**
  - Intercept comms
  - Inject data (SCADA WAN!)





NVIDIA GTX 1080

State-of-the-art... consumer hardware... in 2016...

"**BM**: The researchers found that they were able to decrypt messages from this, using a **very high-powered graphics card** in about a minute."[1]

"**BM**: I suppose all I can say is that **25 years ago the length of this algorithm was probably sufficient to withstand brute-force attacks**.
KZ: You're saying 25 years ago 32 bit would have been secure?
BM: I think so. I can only assume."[1]

"**BM**: I would say it's vulnerable if you happen to be an expert and have some **pretty reasonable equipment**."[1]

[1] Interview between Kim Zetter and Brian Murgatroyd, Chair of ETSI TC TETRA
https://zetter.substack.com/p/interview-with-the-etsi-standards

- Let’s not assume

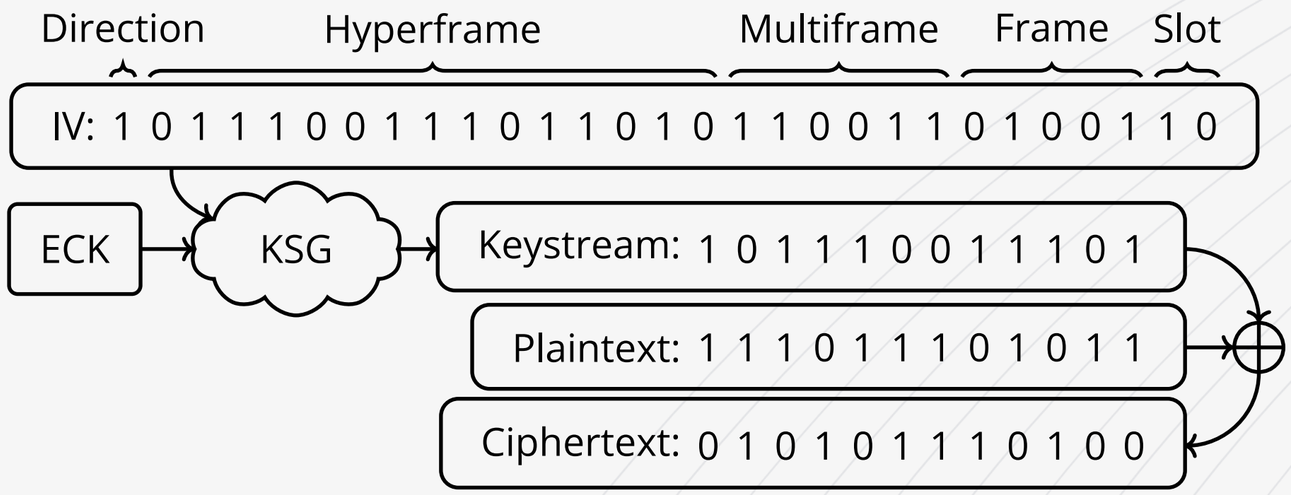
- Let’s not assume
- Let’s not use reasonable equipment

Toshiba Satellite 4010CDS

- Let’s not assume
- Let’s not use reasonable equipment
- Let’s go back to 1998!
  - 266 MHz Pentium II
  - 4.1 billion byte hard disk
  - 32MB SDRAM

Demo: Party like the ‘90s

Air Interface Encryption

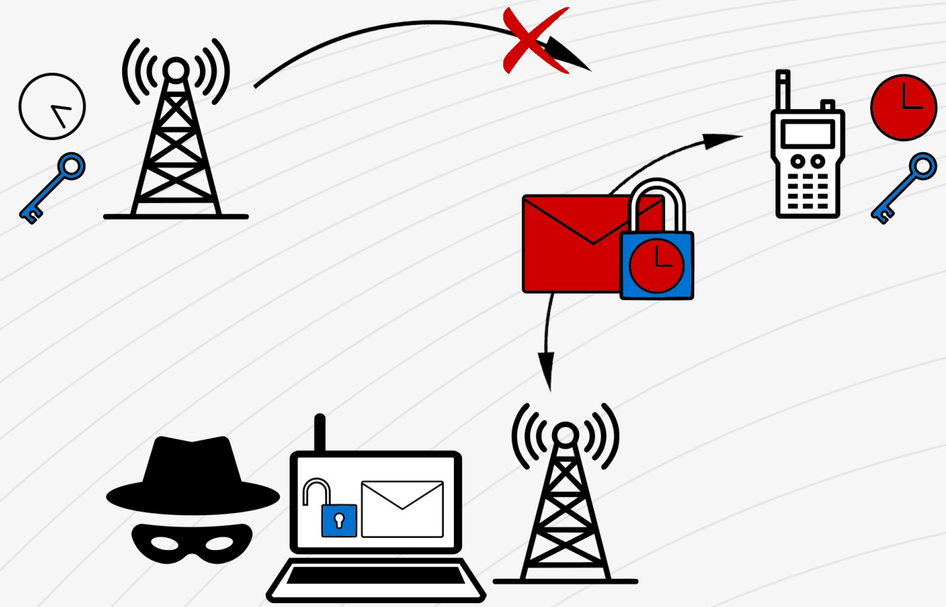


- Air interface signaling is encrypted
- TEAx keystream generators depend on key and on network time
  - Need to guarantee different keystream is used each time
- Network time broadcast in unencrypted, unauthenticated manner
  - SYNC and SYSINFO frames
  - No cryptographic integrity checks
- TETRA messages have no cryptographic auth/integrity guarantee

Attack outline

Attack outline:

- Capture interesting encrypted message at time T
- Target MS (any, with same keys)
- Overpower legitimate signal
- Set MS time to time T
- *Somehow recover keystream for that time*
- …
- Profit

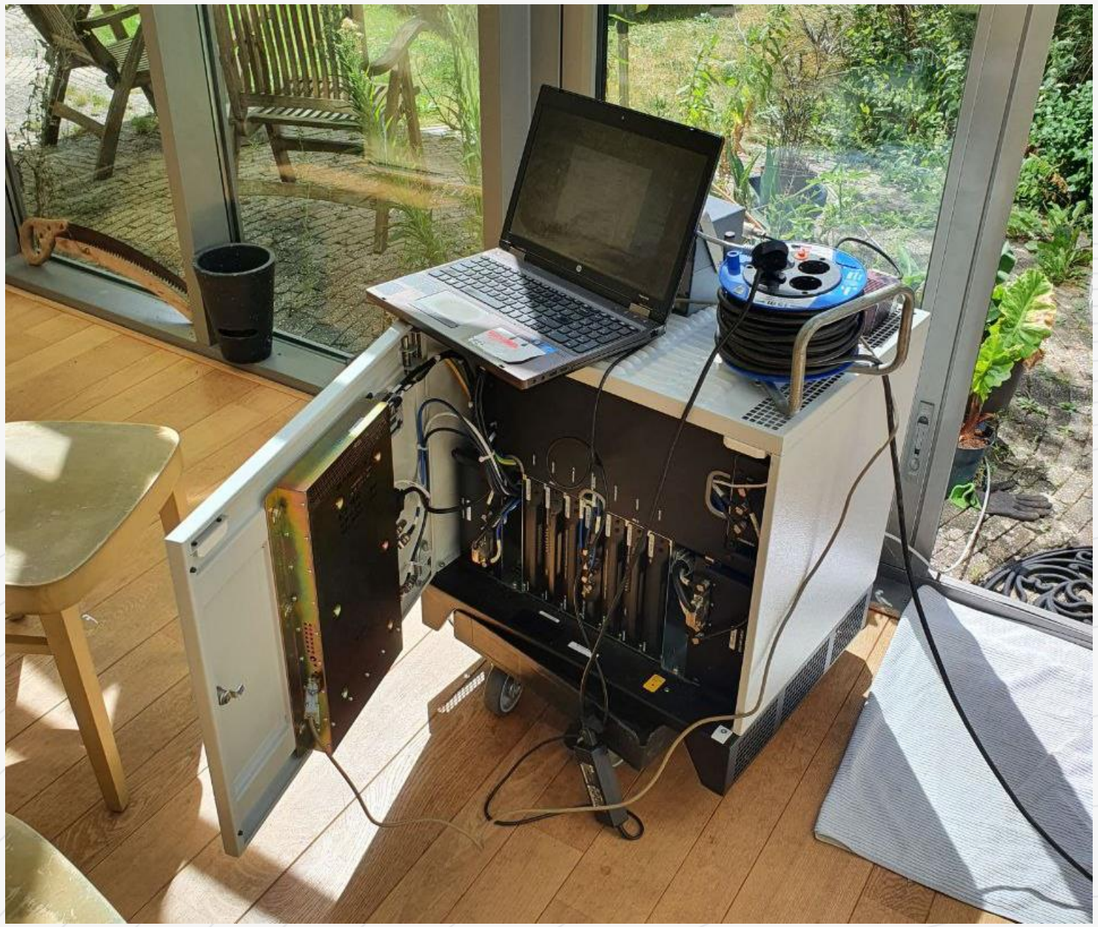


Intermezzo: ETSI

- **“Theoretical attack”**
- **Okay, so, can we have a base station to prove practicality?**
  - Haha lol no
  - More stakeholders responded like this
- **What do we do now?**
  - Implement TETRA infra stack for SDR?
  - Sounds like a lot of work…

Motorola MBTS

- Old
- Heavy
- Clear (no TEA)
- Cool as f*ck
- Turn it into an attack platform!



Demo: CVE-2022-24401

Keystream recovery attack

MBTS vulns

- **Unauthenticated firmwares**
  - Problematic for mission critical comms
  - Allows for key exfiltration or persistent implants
- **Hardcoded undocumented backdoor password**
  - .. at god-mode auth level
- **Crash = drop to debug shell**
  - AKA crash = code execution
- **... Seems there is some kind of industry-wide issue**
  - Similar things in newer BTSes / MSes
  - Is anyone pentesting TETRA equipment?!

Coordinated Vulnerability Disclosure

Timeline

01-2021	• Started work on the RETETRA project
12-2021	• First contact NCSC-NL
01-2022	• First meeting Dutch police
01-2022	• First meeting ETSI
01-2022	• First meeting intelligence community
02-2022	• Detailed preliminary advisory distributed
'22/'23	• Further advisory info & mitigations distributed to stakeholders • Coordinated publication timeline

There is hope…

- **ETSI released TETRA standard revision (TAA2, TEA5-7)**
  - .. Initially, again to be secret
- **ETSI now states they will make all TETRA algorithms public as a result of our research**
- **Assuming they will be fully public*, this would be very welcome!**
  - * = open design criteria, no unexplained constants, no closed vendor reference implementations or integration guidelines, etc.

[1] **"ETSI Releases TETRA Algorithms to Public Domain, maintaining the highest security for its critical communication standard"**
https://www.etsi.org/newsroom/press-releases/2293-etsi-releases-tetra-algorithms-to-public-domain-maintaining-the-highest-security-for-its-critical-communication-standard

Mitigations

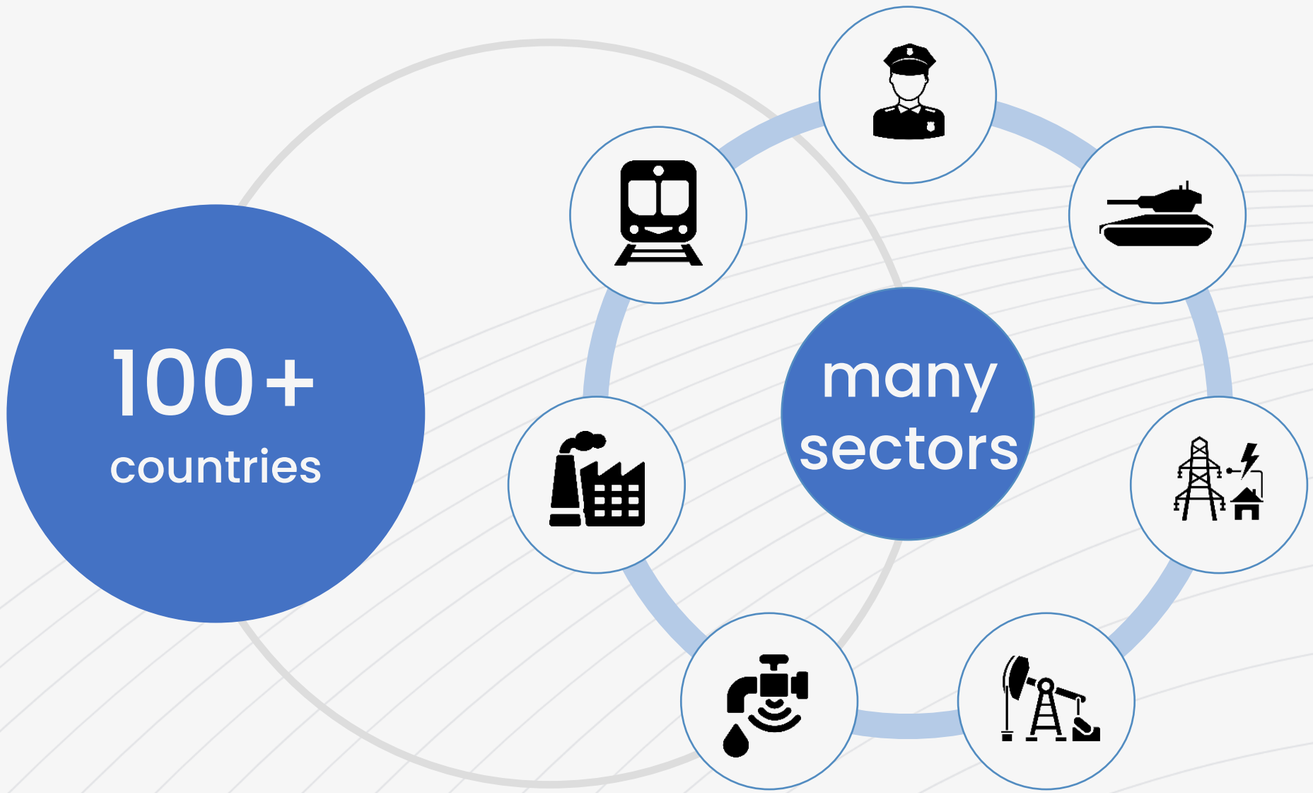
CVE	Description	Recommended Mitigation	Compensating Controls
CVE-2022-24401 CVE-2022-24404	Keystream recovery attack	• Firmware updates • E2E • (data) TLS / IPsec	• Renew keys frequently • Risk assessment, adjust OPSEC
CVE-2022-24402	TEA1 backdoor	• TEA2 • E2E • (data) TLS / IPsec	• Assume TEA1 == cleartext • Risk assessment, adjust OPSEC
CVE-2022-24403	Deanonymization attack	• Migrate to TAA2	• Risk assessment, adjust OPSEC
CVE-2022-24400	DCK key pinning attack	• Firmware updates • Migrate to TAA2	• Disable radios with unacceptable FW update rollout timelines

Note of warning

- Not all vendors have been properly informing their customers!
- As of October 2023, we have received *multiple* reports of vendors continuing to recommend TEA1, falsely claim it can be ‘patched’ without migration, or downplay impact of TETRA:BURST vulnerabilities
- We have also received reports of *incomplete* patches leaving risks unaddressed
- Please feel free to reach out to us when in doubt

Conclusion

- First public, in-depth TETRA security analysis (after 20+ years)
- Secret crypto algorithms reverse-engineered
- Multiple vulnerabilities uncovered (incl. backdoor)
- Implications for voice, data, and SCADA
- Patches available for some issues, mitigations for others
- Lots of work still to be done for asset owners!



Questions?

Social

-

Web

- midnightblue.nl
- tetraburst.com

Contact

- c.meijer@midnightblue.nl
- w.bokslag@midnightblue.nl
- j.wetzels@midnightblue.nl