black hat USA 2023

AUGUST 9-10, 2023

BRIEFINGS

Apple PAC, Four Years Later

Reverse Engineering the Customized Pointer Authentication Hardware Implementation on Apple M1

Zechao Cai (@Zech4o)

Whoami

- Zechao/Zachary Cai @Zech4o -

Master Student at Zhejiang University

Focus on
- OS Security
- Reverse Engineering
- Virtualization

Contributors

Jiaxun Zhu (@svnswords):

- Student at **Zhejiang University**
- Member of **AAA** CTF Team
- Focus on *OS security and Android Hook
- Building **M1 macOS** fuzzing framework and unlimited debugger

Wenbo Shen:

- ZJU100 Professor at **Zhejiang University**
- Focus on operating system security, software supply chain security, and container security
- Won three distinguished paper awards (**NDSS 16**, **AsiaCCS 17**, **ACSAC 22**)

Yutian Yang:

- Student at **Zhejiang University**
- Working toward a Ph.D. degree
- Focus on OS kernel security and static program analysis for bug detection
- Won **ACSAC 22** distinguished paper award

Yu Wang:

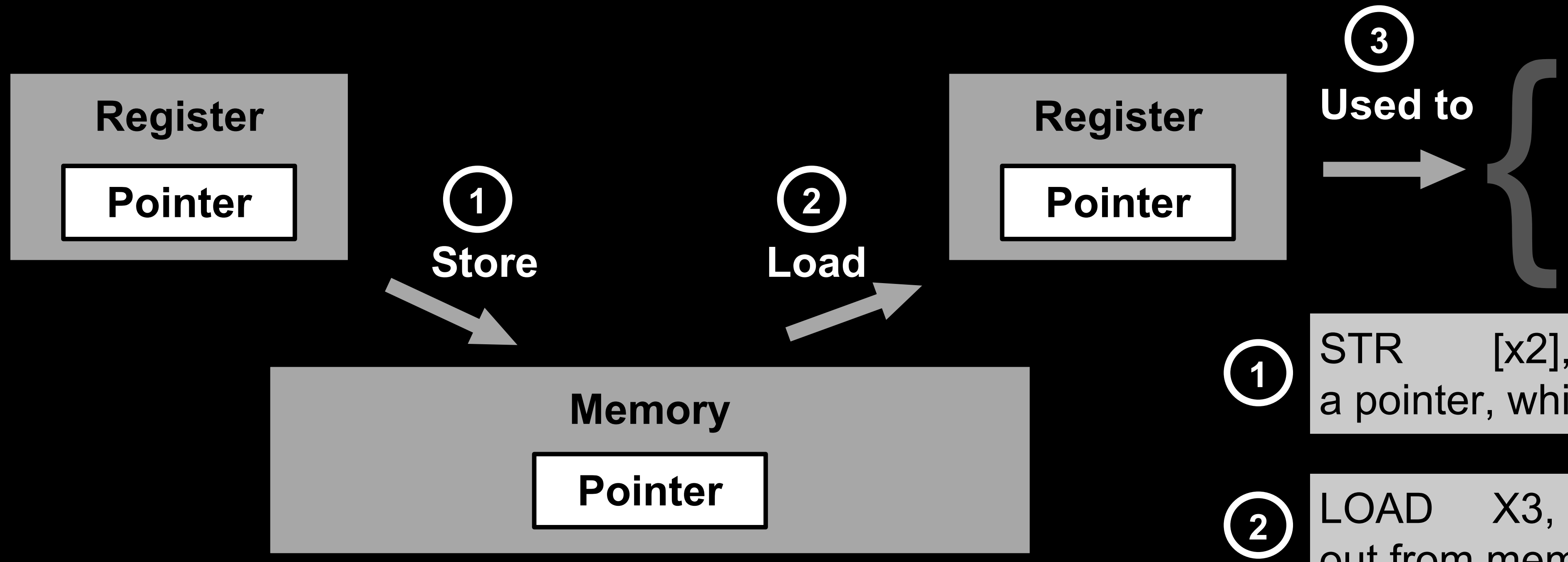
- Founder of **CyberServal** Co., Ltd.
- Focus on kernel architecture, device driver development, rootkit/anti-rootkit solutions to vulnerability hunting and exploitation
- Spoken at **Black Hat**, **DEF CON** and other conferences

Talk Roadmap

- About Pointer Authentication (PAC)
  - What is PAC and Current State of Apple PAC Research
- How I Reverse Engineer it
  - Two Main Challenges
    - Apple-spec Sysreg
    - PAC Key Protection
- Our Findings on Apple PAC Hardware
  - How does Apple achieve Cross-domain Attack Mitigation

Let's look at a basic memory attack

A Simple Example of Pointer's Life Cycle



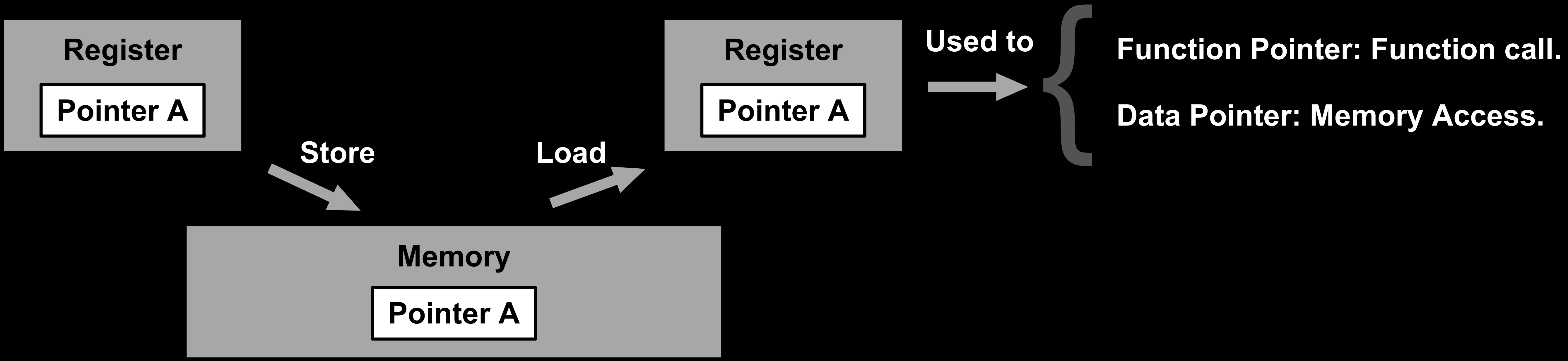
Function Pointer: Function call.

Data Pointer: Memory Access.

1. STR [x2], X1 ; The value in X1 register is a pointer, which is stored into memory
2. LOAD X3, [x2] ; The pointer is then loaded out from memory
3. BL X3; Function call
   or
   LOAD X4, [X3]; Memory Access

Let’s look at a basic memory attack

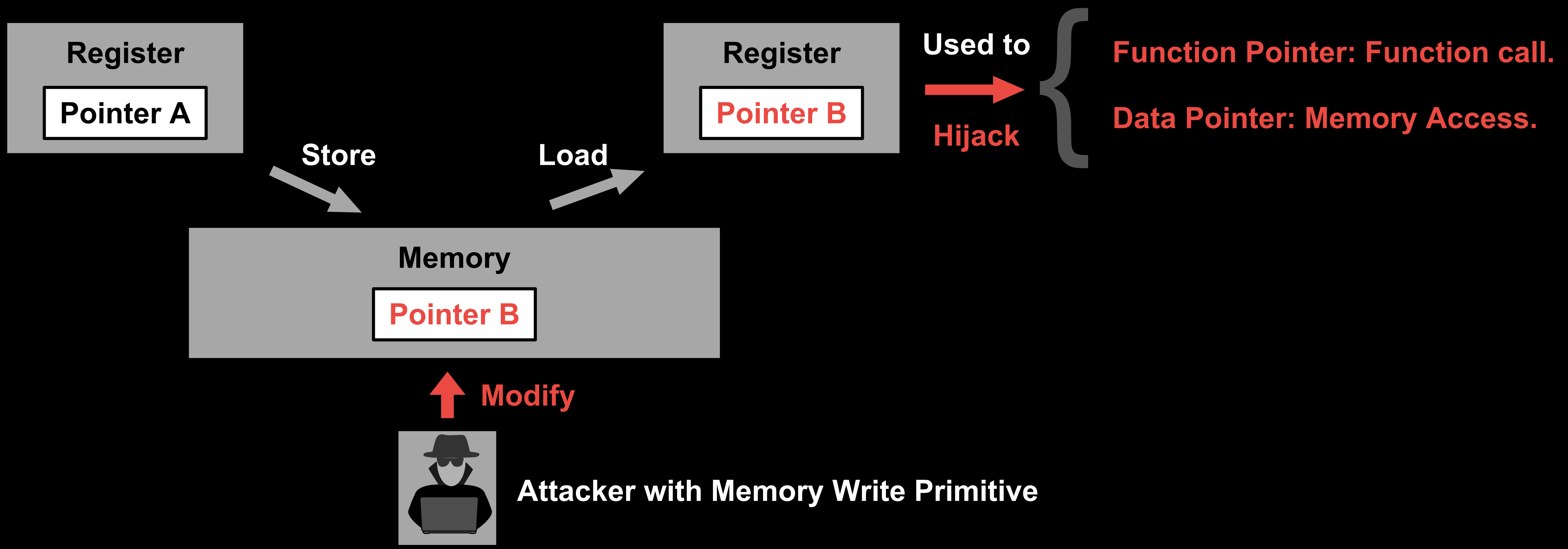
A Simple Example of Memory Corruption Attack



Attacker with Memory Write Primitive

Let's look at a basic memory attack

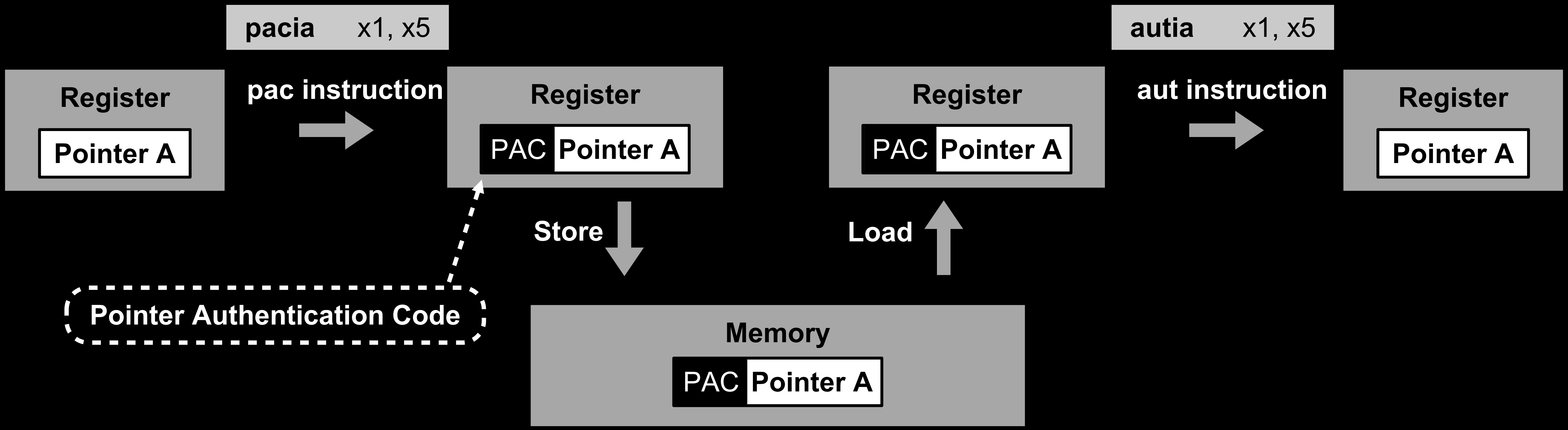
A Simple Example of Memory Corruption Attack



How Apple mitigates this Attack

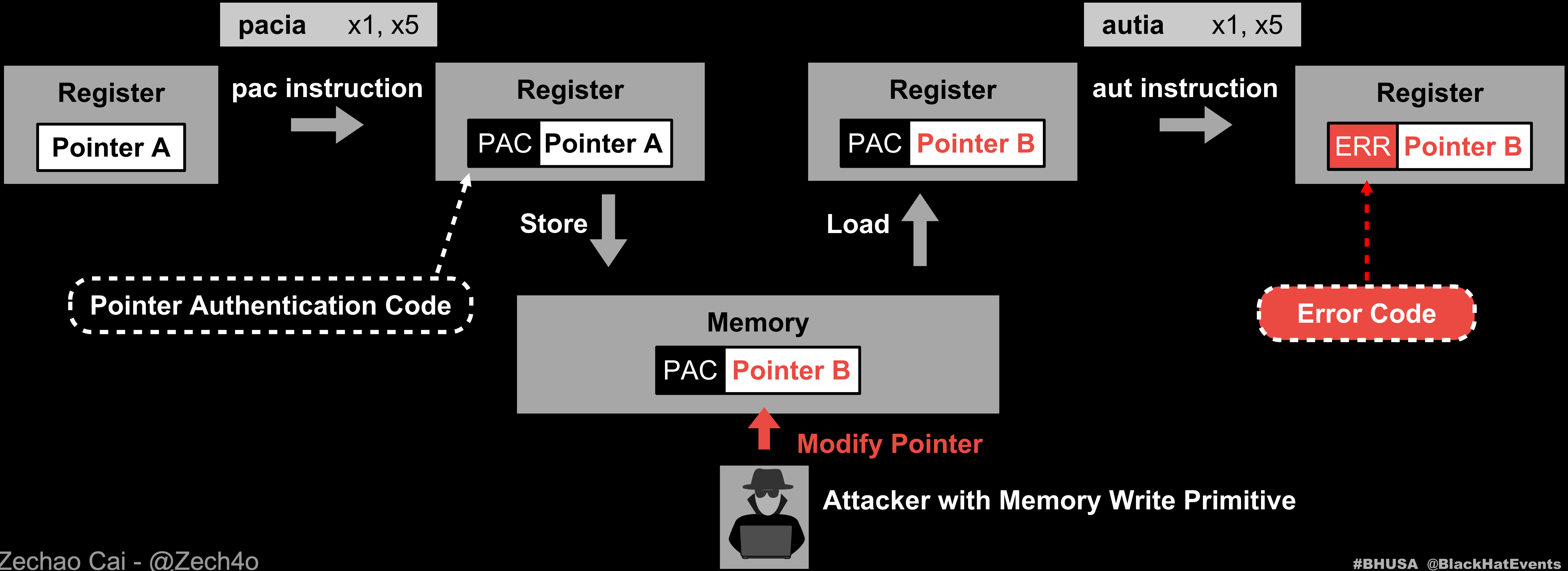
What is Pointer Authentication (PAC)

Basic Usage of Pointer Authentication



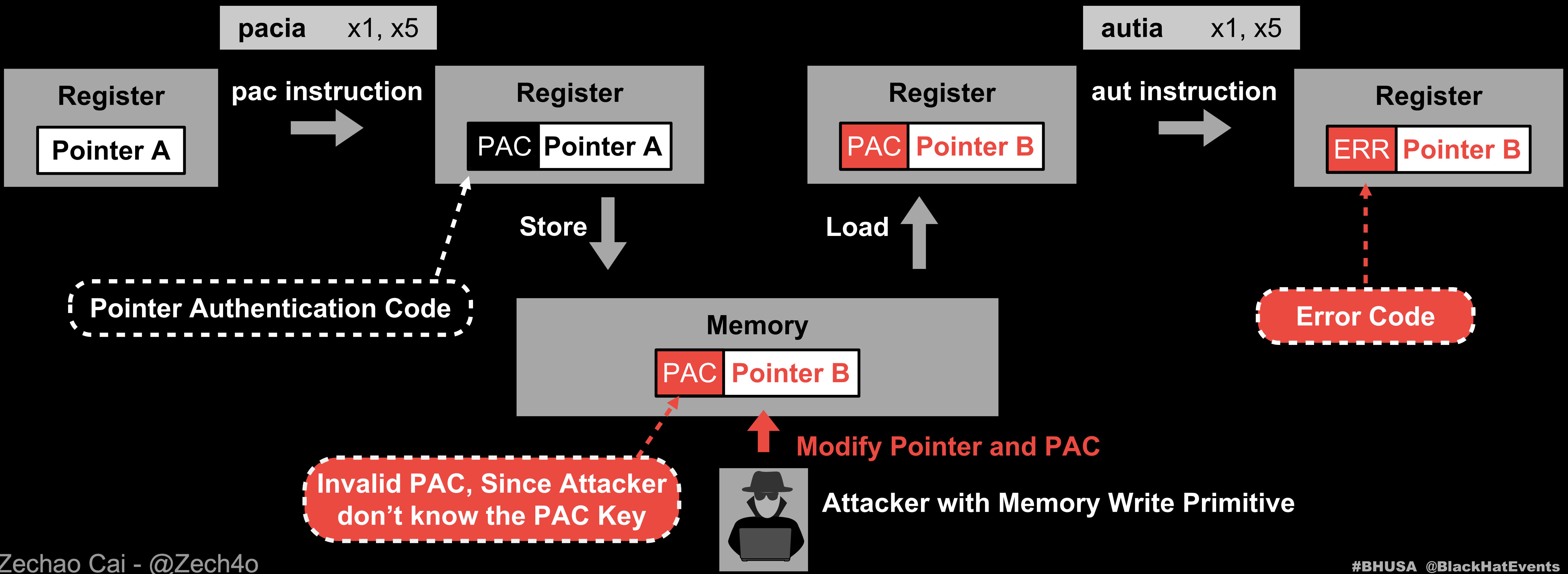
What is Pointer Authentication (PAC)

Basic Usage of Pointer Authentication



What is Pointer Authentication (PAC)

Basic Usage of Pointer Authentication



What is Pointer Authentication (PAC)

ARMv8.3 Specification

Five **128-bit** PAC Keys (Each Key is made up by **two 64-bit** System registers)

What is Pointer Authentication (PAC)

ARMv8.3 Specification

Five **128-bit** PAC Keys (Each Key is made up by **two 64-bit** System registers)

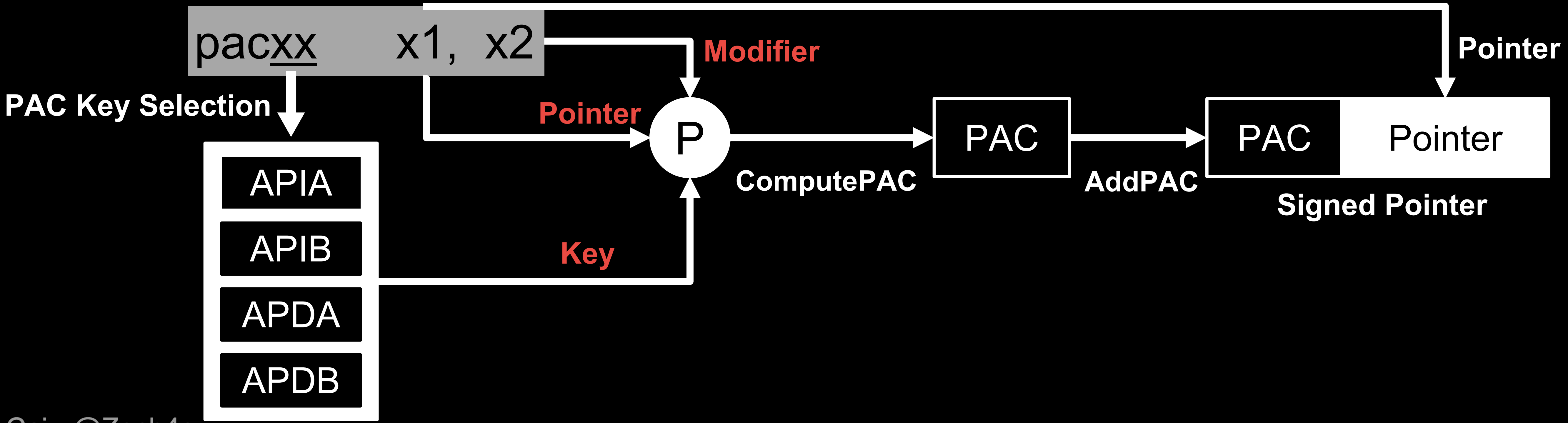
- APIA/IB/DA/DB for Pointer Signing (I: instruction; D: Data)
- APGA for Signature Generation (G: General)

What is Pointer Authentication (PAC)

ARMv8.3 Specification

Five **128-bit** PAC Keys (Each Key is made up by **two 64-bit** System registers)

- APIA/IB/DA/DB for Pointer Signing (I: instruction; D: Data)
- APGA for Signature Generation (G: General)



What is Pointer Authentication (PAC)

ARMv8.3 Specification

Five **128-bit** PAC Keys (Each Key is made up by two 64-bit Sysreg)

- APIA/IB/DA/DB for Pointer Signing (I: instruction; D: Data)
- APGA for Signature Generation (G: General)
- **Only one set of PAC Keys for Exception Level 0/1/2**

What is Pointer Authentication (PAC)

ARMv8.3 Specification

Five **128-bit** PAC Keys (Each Key is made up by two 64-bit Sysreg)

- APIA/IB/DA/DB for Pointer Signing (I: instruction; D: Data)
- APGA for Signature Generation (G: General)
- Only one set of PAC Keys for Exception Level 0/1/2

One Control Register - SCTLR_EL1

- **Per-Key Switches**
- **EnIA/EnIB/EnDA/EnDB bits to enabled/disable pac instruction**

Apple PAC

Since A12 (iPhone XS, 2018)

Feature	A10	A11, S3	A12, S4	A13, S5	A14, A15, S6, S7	M1 Family
Kernel Integrity Protection	✅	✅	✅	✅	✅	✅
Fast Permission Restrictions		✅	✅	✅	✅	✅
System Coprocessor Integrity Protection			✅	✅	✅	✅
Pointer Authentication Codes			✅	✅	✅	✅
Page Protection Layer		✅	✅	✅	✅	See Note below.

Current Research State of Apple PAC

Most of Research works focus on Software/PAC Bypass

Software (Kernel)

Examining Pointer Authentication on the iPhone XS
- Brandon Azad (Google Project Zero)

Attacking iPhone XS Max (Black Hat USA 2019)
- Tielei Wang and Hao Xu (Team Pangu)

2PAC 2Furious: Envisioning an iOS compromise in 2019
- Macro Grassi and Liang Chen KEEN Security Lab

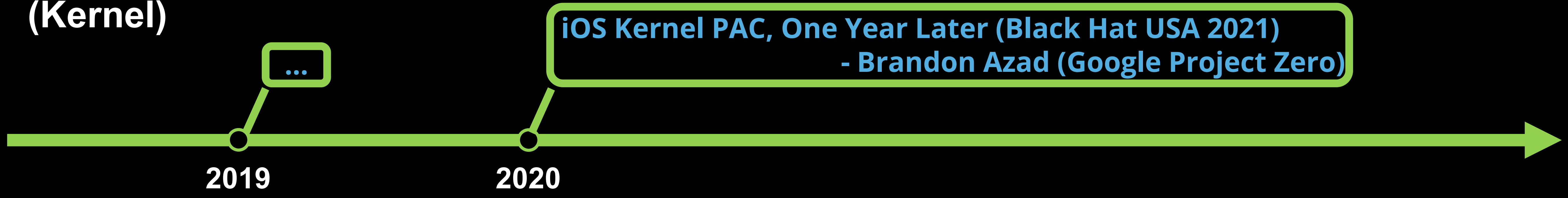
2019

Hardware

Current Research State of Apple PAC

Most of Research works focus on Software/PAC Bypass

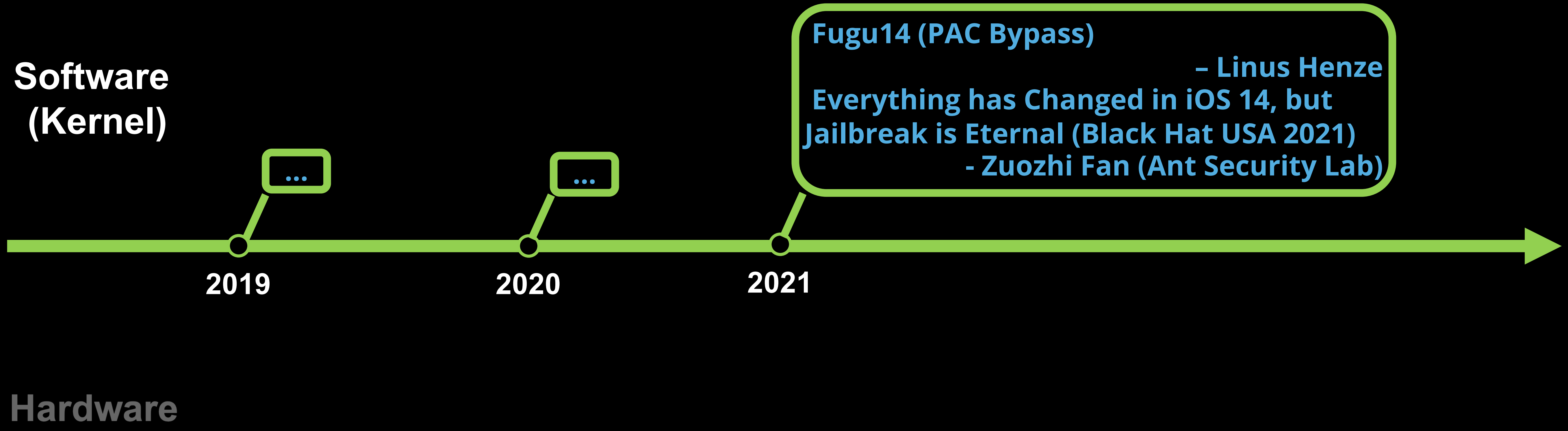
Software (Kernel)



Hardware

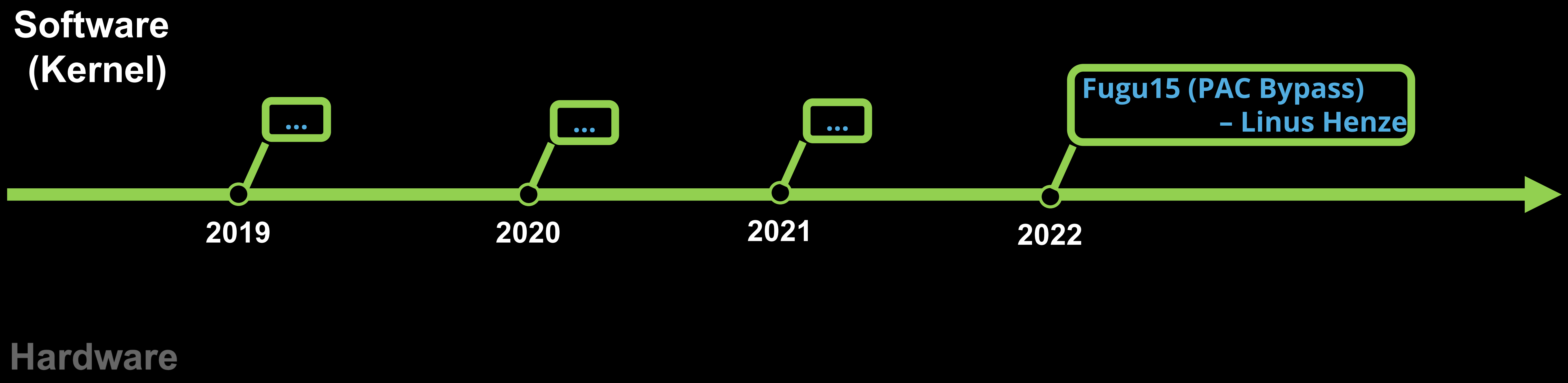
Current Research State of Apple PAC

Most of Research works focus on Software/PAC Bypass



Current Research State of Apple PAC

Most of Research works focus on Software/PAC Bypass

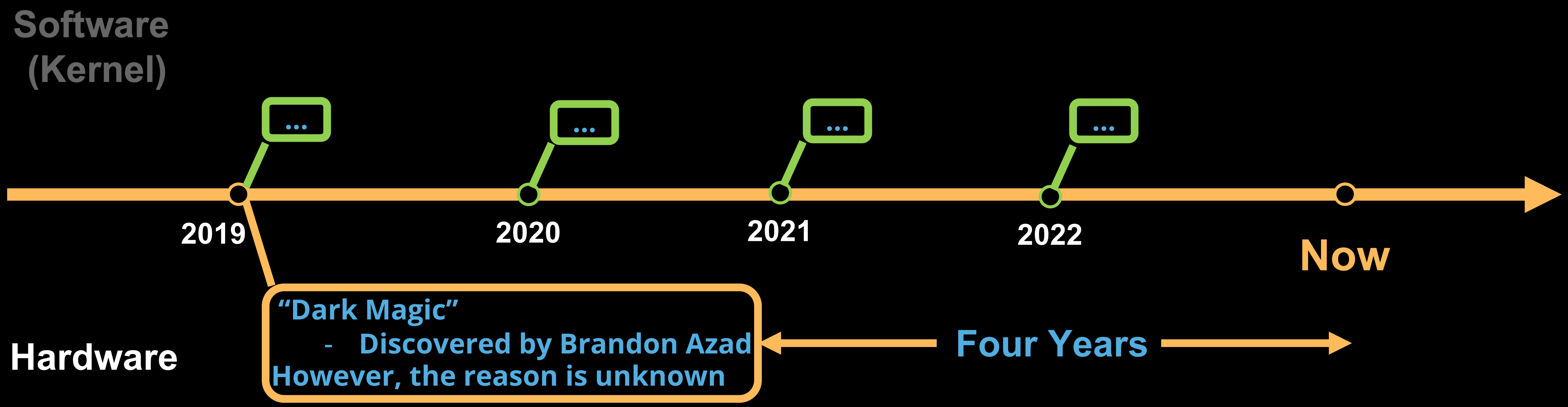


Current Research State of Apple PAC

Most of Research works focus on Software/PAC Bypass

Brandon Azad found that Apple customized the PAC hardware.

But the implementation behind the "Dark Magic" remains unknown.



Recap of "Dark Magic"

Cross-domain Attack

Pointer Substitution Attack across different domains

*ARM PAC does not provide hardware isolation

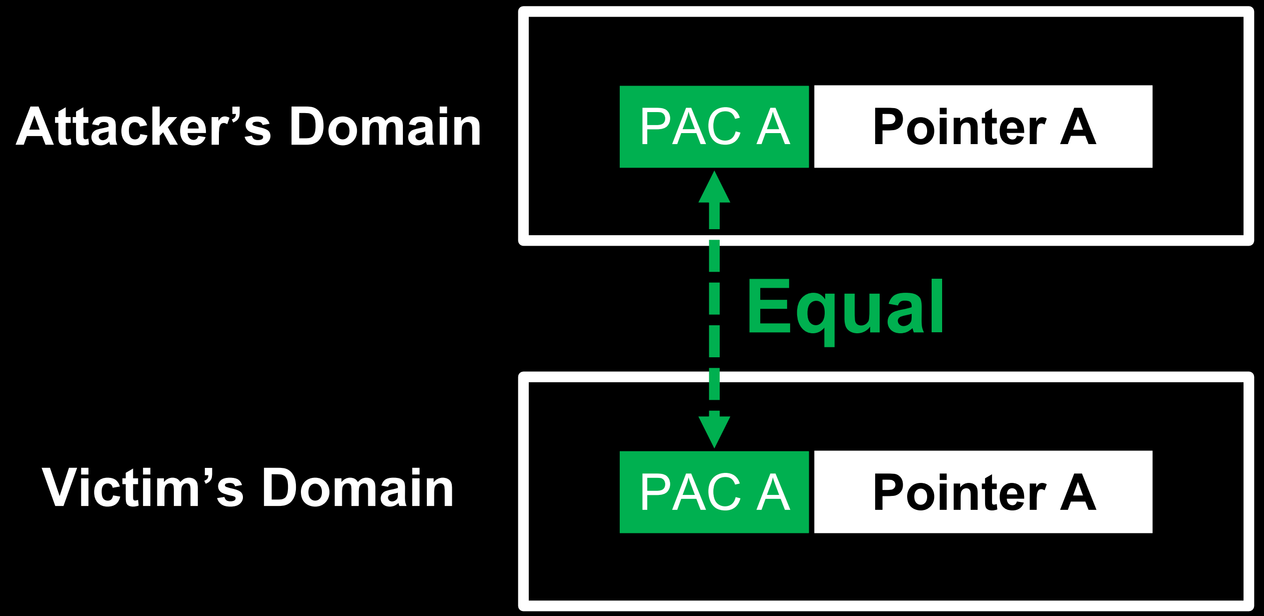


Fig. Signing Pointers with same inputs (Key Type, Key Value, Pointer, Modifier) in different domains

Recap of "Dark Magic"

Cross-domain Attack

Pointer Substitution Attack across different domains

*ARM PAC does not provide hardware isolation

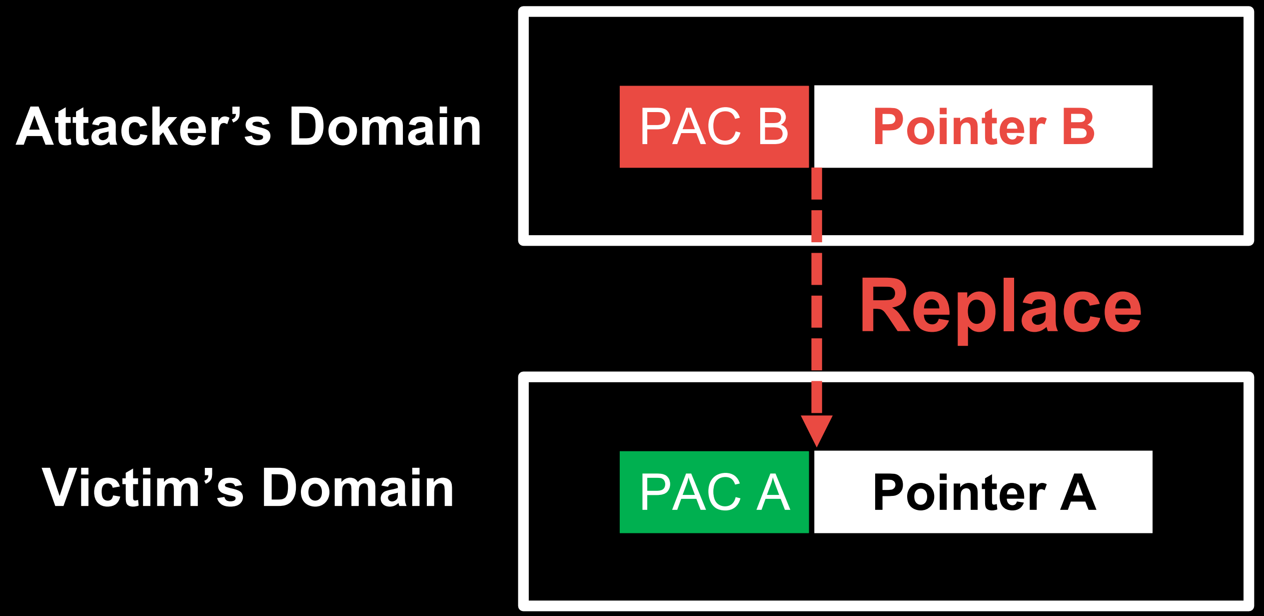


Fig. Hijack the Control/Data flow in victim's domain by replacing the pointer without being detected

Recap of "Dark Magic"

Cross-domain Attack

Pointer Substitution Attack across different domains

*ARM PAC does not provide hardware isolation

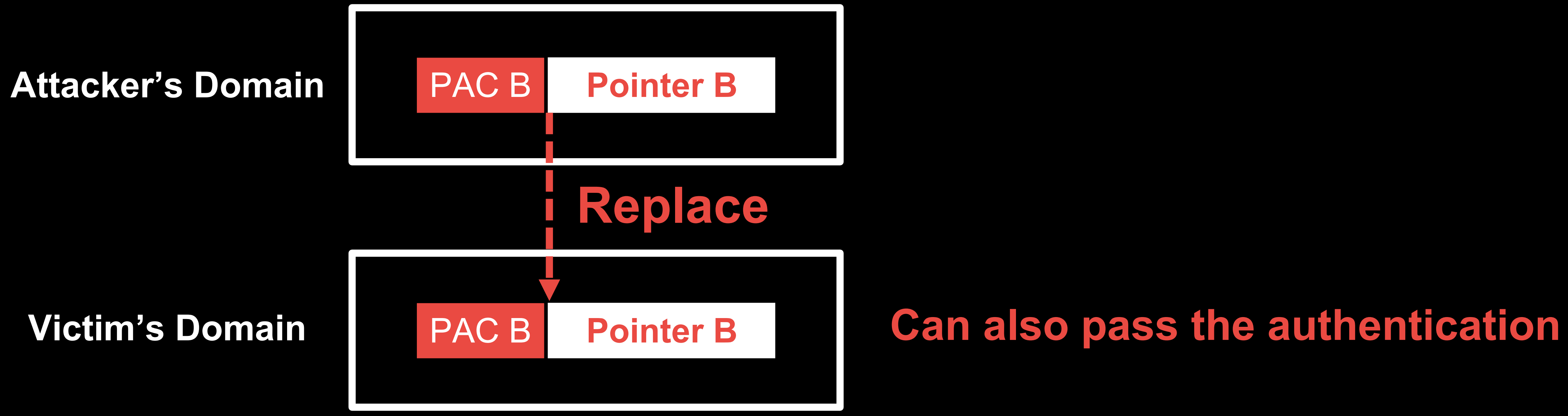
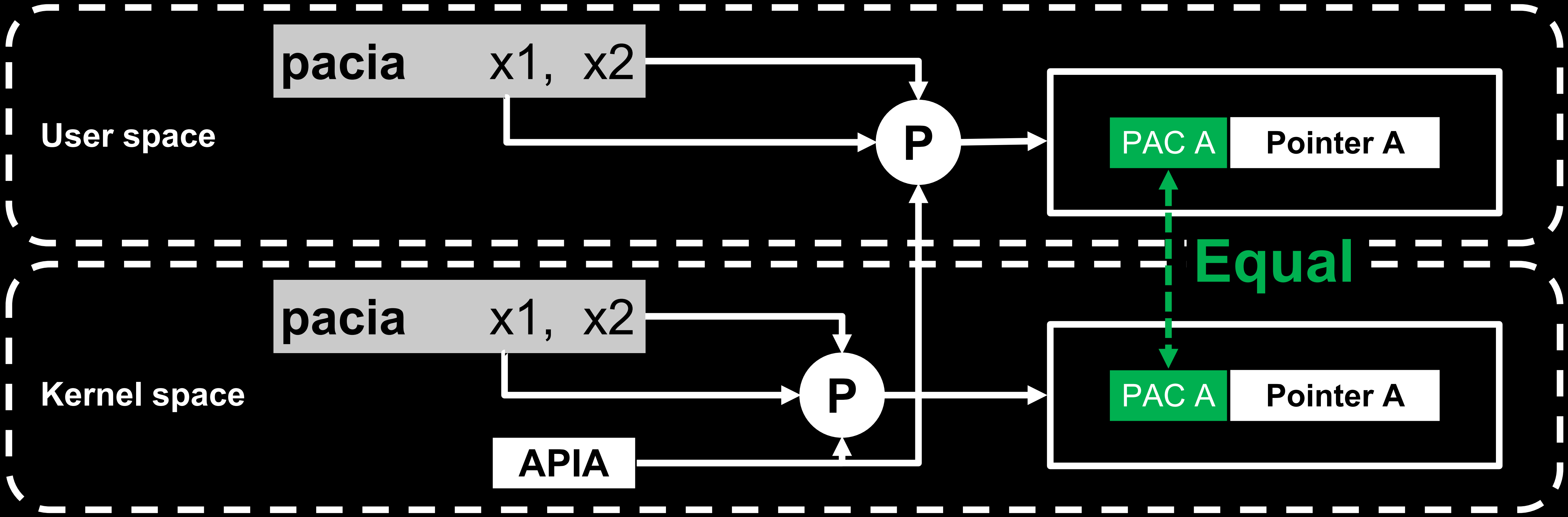


Fig. Hijack the Control/Data flow in victim's domain by replacing the pointer without being detected

Recap of "Dark Magic"

e.g. Cross-EL Attack

Attacker tries to generates a signed kernel pointer in user space

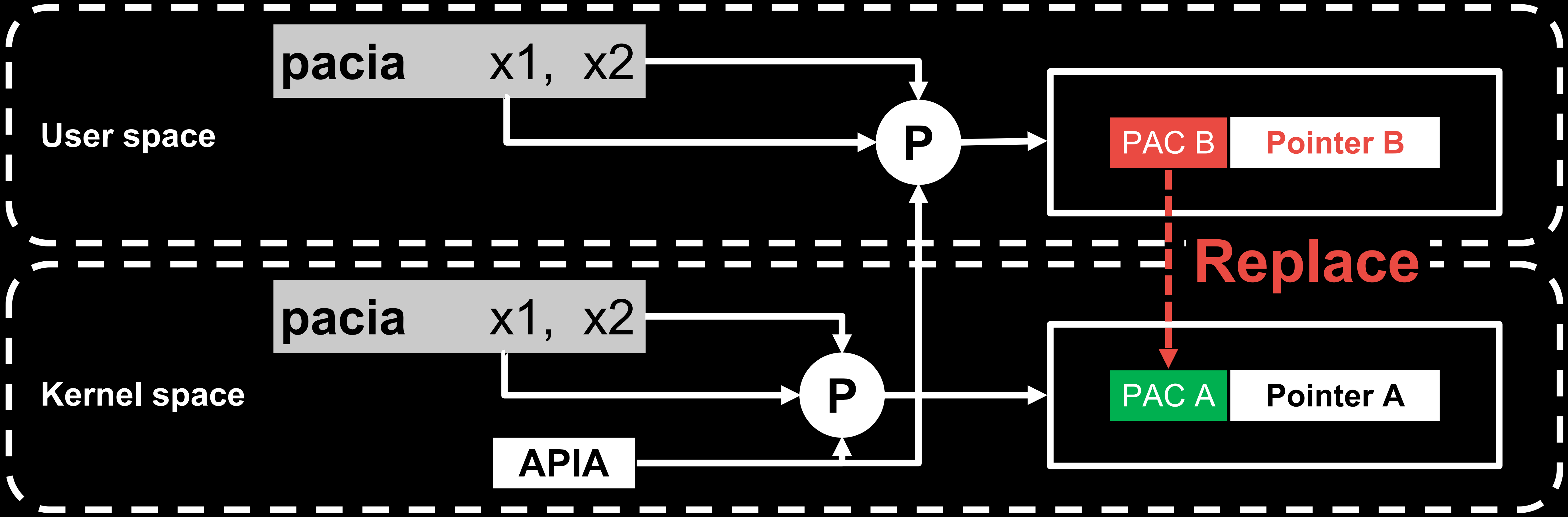


ARM PAC

Recap of “Dark Magic”

e.g. Cross-EL Attack

Attacker tries to generates a signed kernel pointer in user space

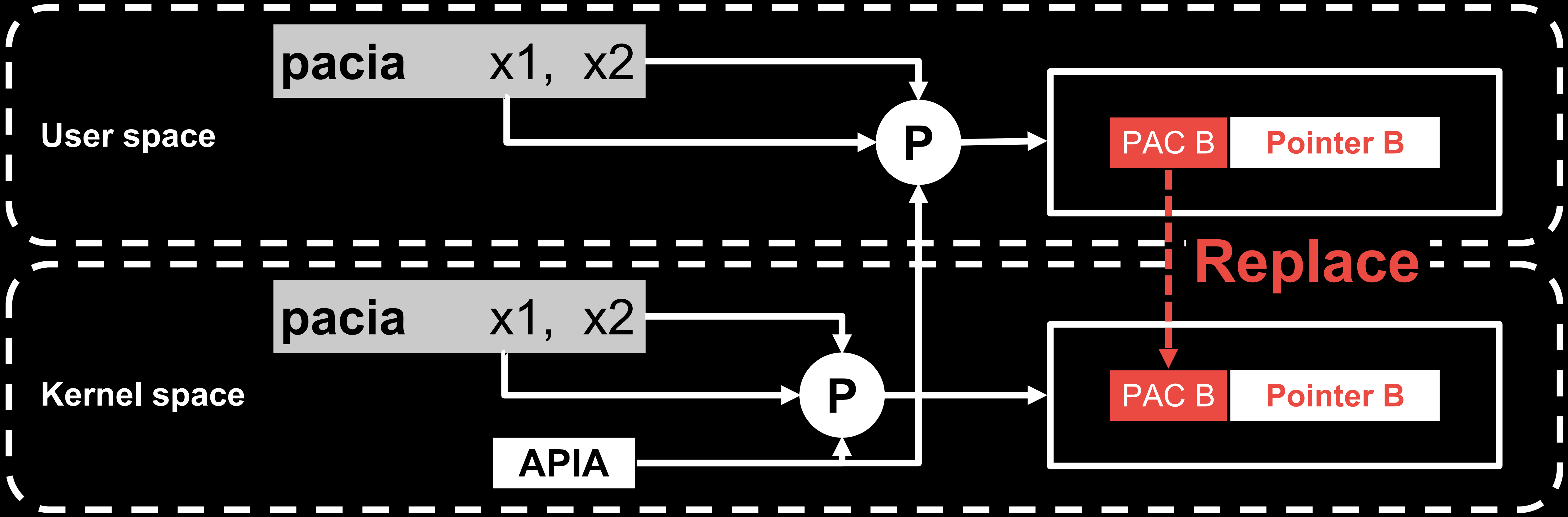


ARM PAC

Recap of "Dark Magic"

e.g. Cross-EL Attack

Attacker tries to generates a signed kernel pointer in user space



ARM PAC

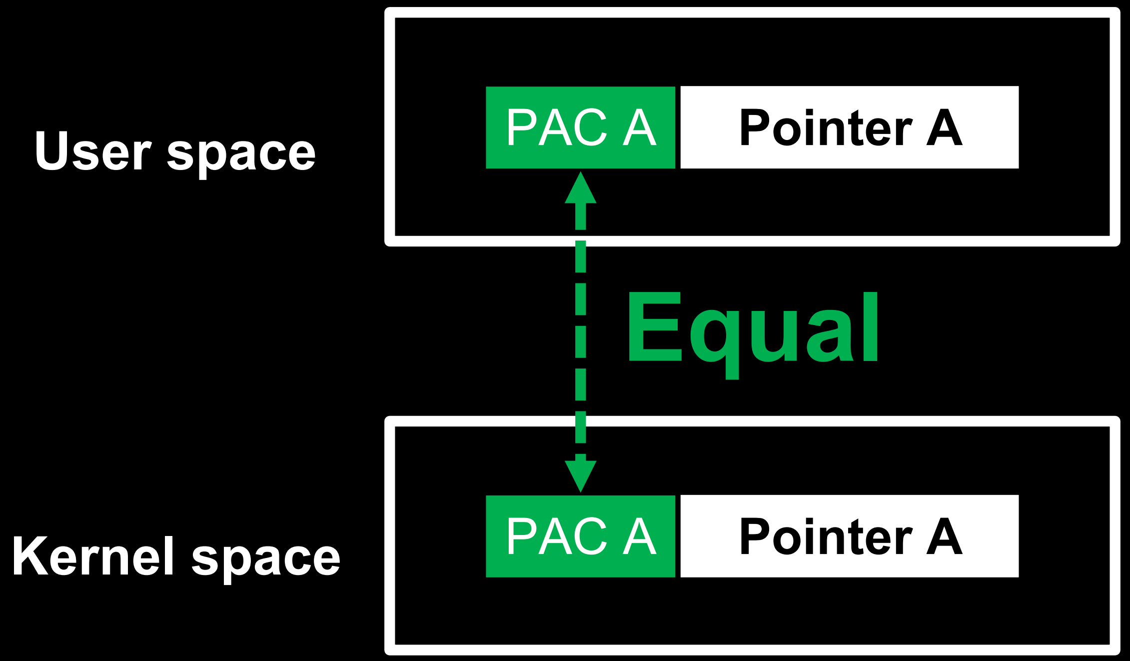
Recap of “Dark Magic”

e.g. Cross-EL Attack

Attacker tries to generates a signed kernel pointer in user space

Existing works mitigate cross-EL Attack by

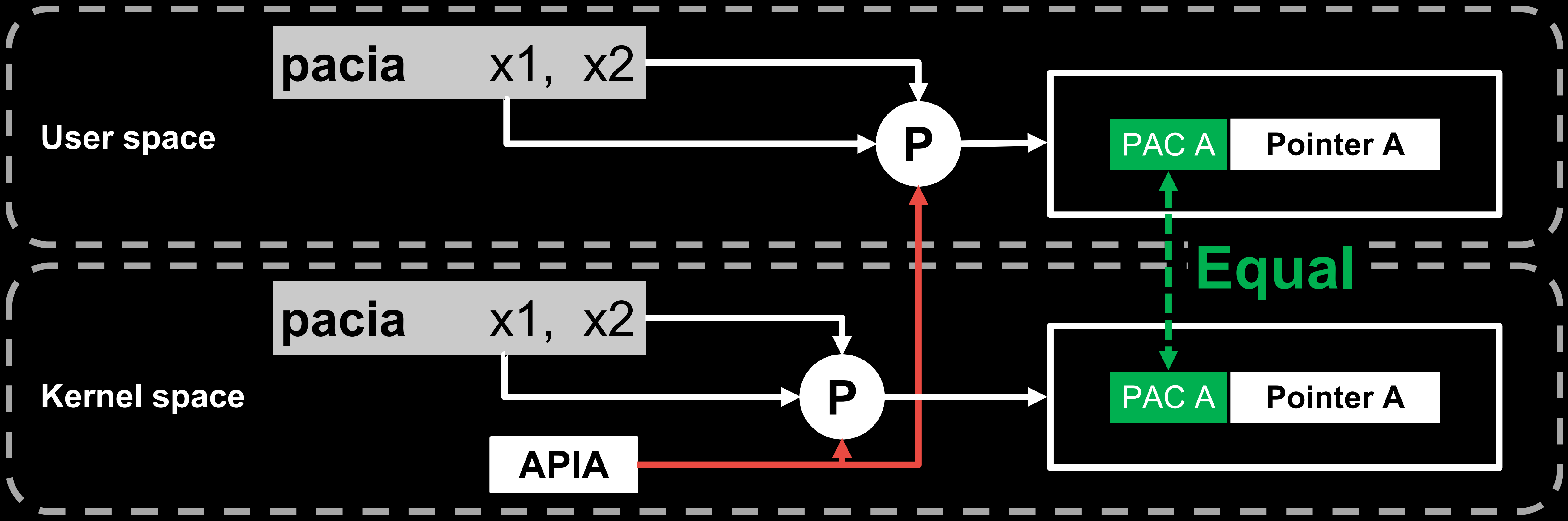
- Maintaining different key values
- Disabling user space PAC



Recap of "Dark Magic"

1. Cross-EL Attack Mitigation on Apple Silicon

There is no key switching operation in the XNU kernel

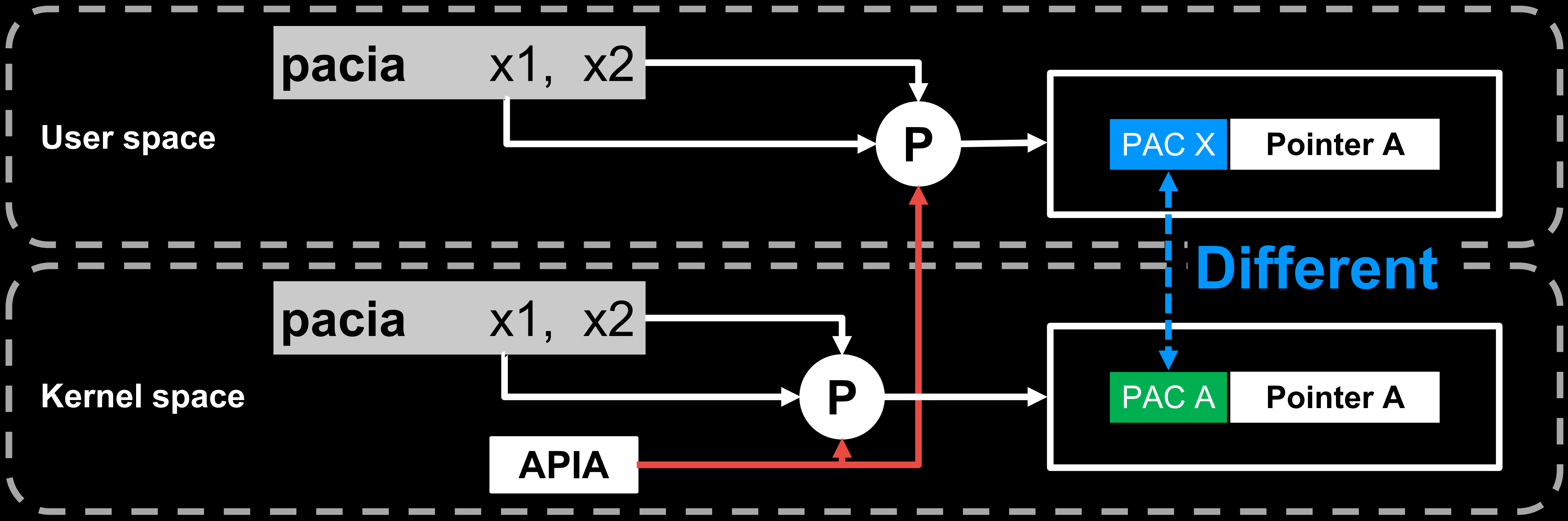


ARM PAC

Recap of “Dark Magic”

1. Cross-EL Attack Mitigation on Apple Silicon

There is no key switching operation in the XNU kernel

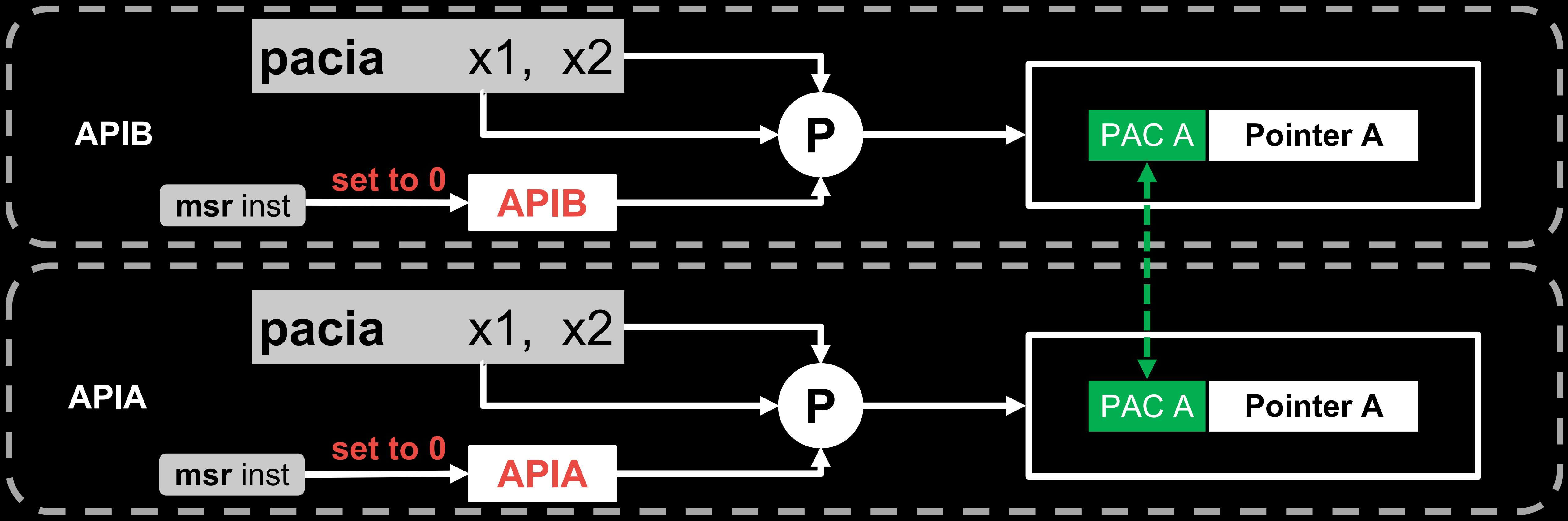


Apple PAC

Recap of "Dark Magic"

2. Cross-Key Attack Mitigation on Apple Silicon

Set up **different key using the same values**

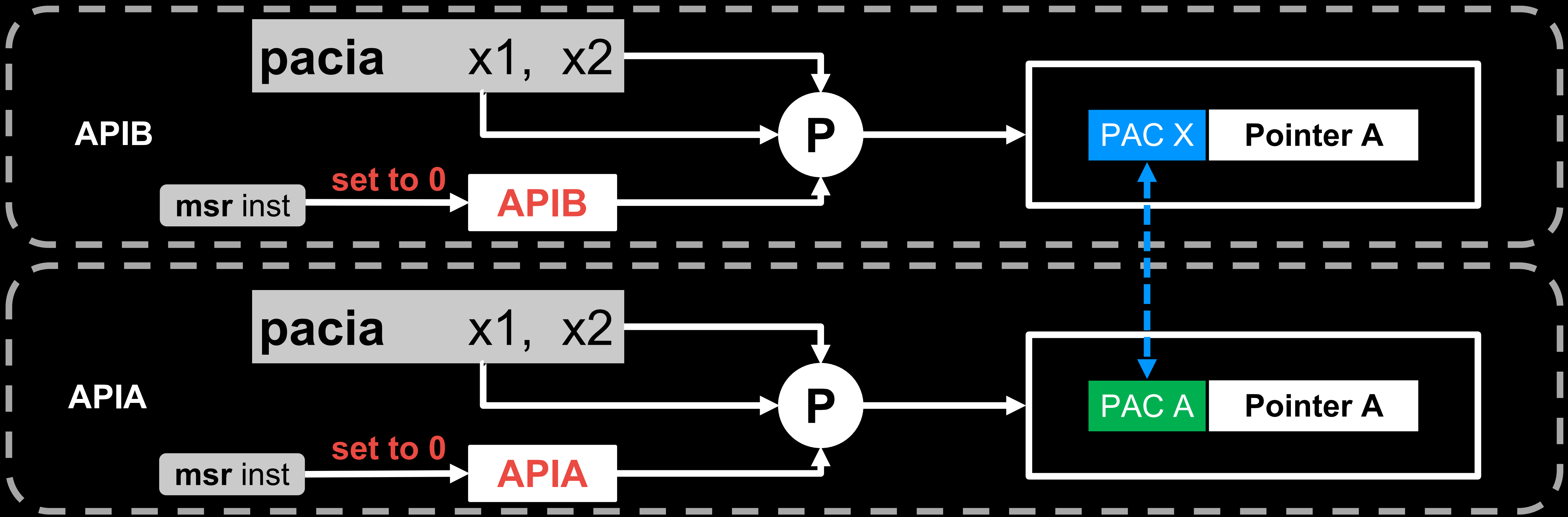


ARM PAC

Recap of “Dark Magic”

2. Cross-Key Attack Mitigation on Apple Silicon

Set up **different key using the same values**

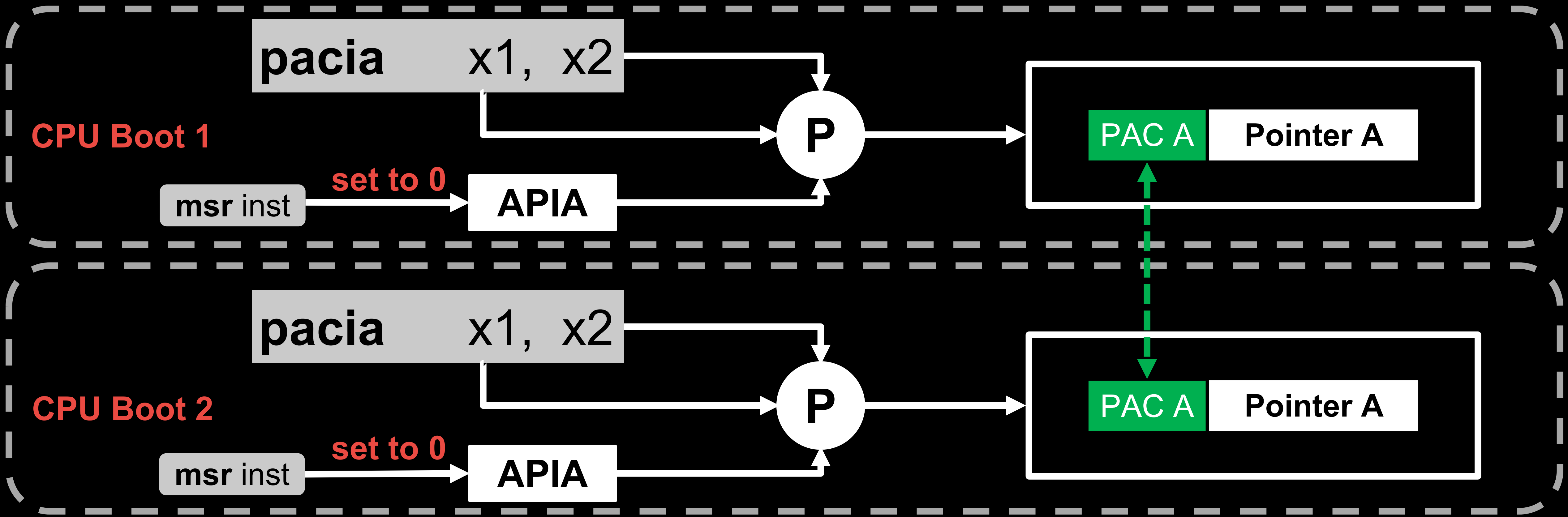


Apple PAC

Recap of "Dark Magic"

3. Cross-Boot Attack Mitigation on Apple Silicon

Set up **the key with the static value after each CPU boot**

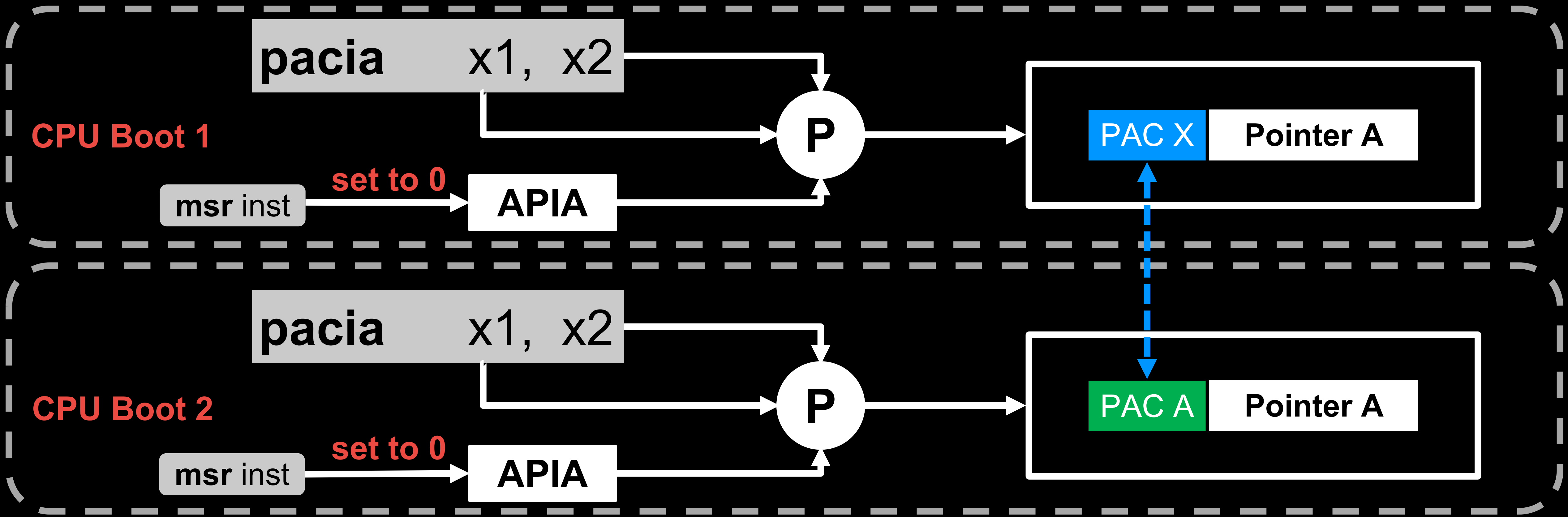


ARM PAC

Recap of "Dark Magic"

3. Cross-Boot Attack Mitigation on Apple Silicon

Set up <u>the key with the static value after each CPU boot</u>

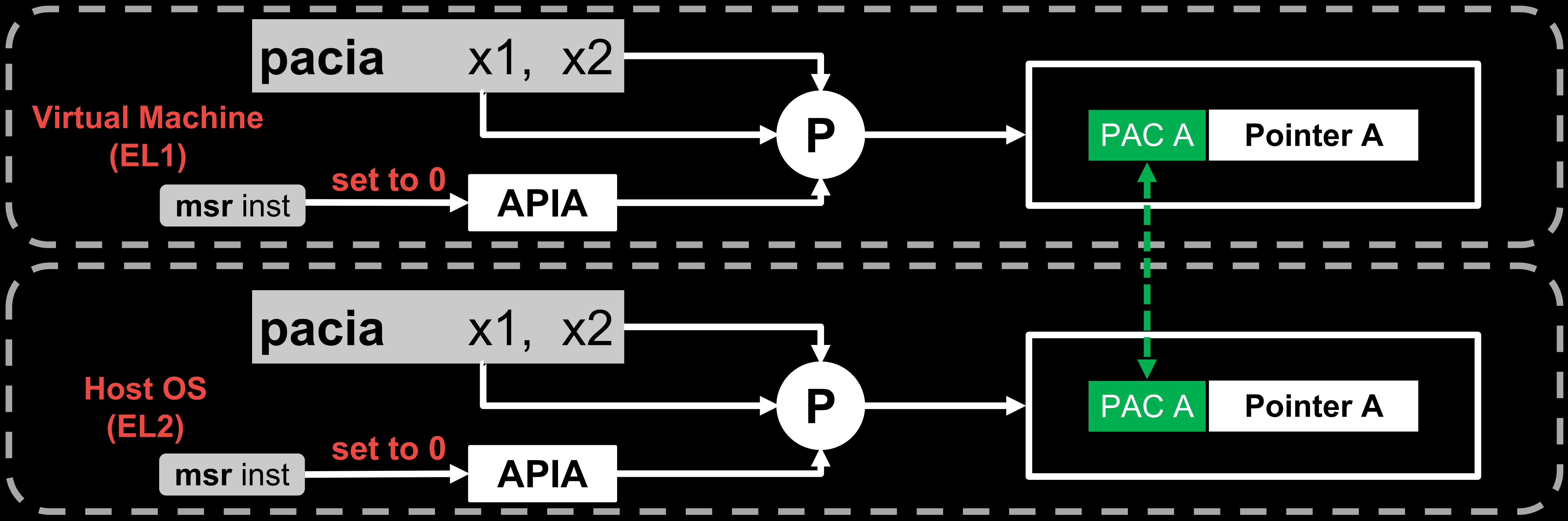


Apple PAC

Recap of "Dark Magic"

4. Cross-VM Attack Mitigation on Apple Silicon (Apple M1)

Set up **the keys with the same key values in VM and Host**

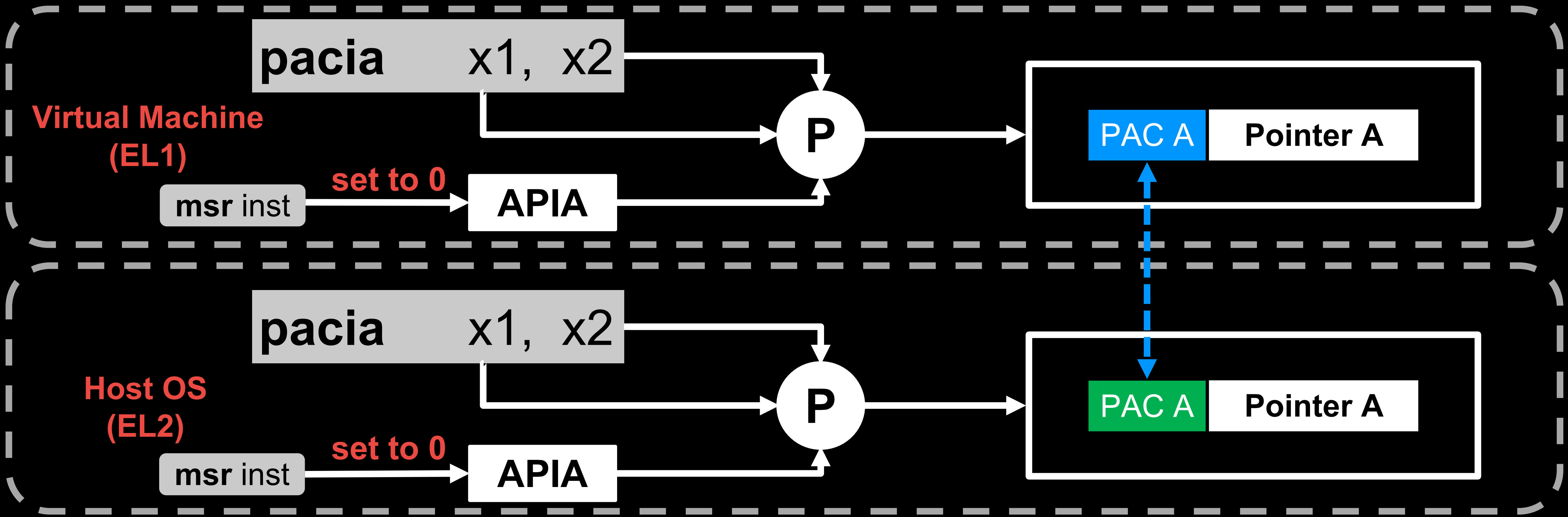


ARM PAC

Recap of "Dark Magic"

4. Cross-VM Attack Mitigation on Apple Silicon (Apple M1)

Set up the keys with the same key values in VM and Host



Apple PAC

"Dark Magic" – My Main Research Motivation

Apple implements Cross-domain Attack Mitigation without software support.

How does Apple customized the PAC hardware?

"Dark Magic" – Our Main Research Motivation

Apple implements Cross-domain Attack Mitigation without software support.

How does Apple customized the PAC hardware?

You will know how Apple implements it after this talk.

How I Reverse Engineer Apple PAC

Zechao Cai - @Zech4o

Basic idea

Basic idea

Change CPU States and See what happens

Basic idea

Change CPU States and See what happens

Set System Register

Step 1

Basic idea

Change CPU States and See what happens

Set System Register

Run Instructions

Step 1

Step 2

How I Reverse Engineer

Challenge 1

- What are the system registers we want to set?
- Apple introduced undocumented system registers

Set System Register

How I Reverse Engineer

Challenge 1

- What are the system registers we want to set?
- Apple introduced undocumented system registers

Set System Register

Run Instructions

Challenge 2

- How to read the PAC key
- Apple introduce hardware PAC key protection

How I Reverse Engineer

Task 1

- Identify Apple-spec PAC-related undocumented system registers

Set System Register

How I Reverse Engineer

Task 1

- Identify Apple-spec PAC-related undocumented system registers

Set System Register

Run Instructions

Task 2

Bypass Apple-spec hardware PAC key protection

Task 1. Apple-spec PAC system register identification

Task 1. Apple-spec PAC system register identification

System Register

Registers for configuring the CPU feature

Accessed by ‘msr’ (write) and ‘mrs’ (read) instructions

e.g. TTBR1_EL1, Translation Table Base Register 1 (EL1)

```
msr    TTBR1_EL1,  X1
```

Task 1. Apple-spec PAC system register identification

TTBR1_EL1 is a register.

```
msr     TTBR1_EL1,  X1
```

Task 1. Apple-spec PAC system register identification

~~TTBR1_EL1 is a register.~~ ✖

```
msr    TTBR1_EL1, X1
```

Task 1. Apple-spec PAC system register identification

~~TTBR1_EL1 is a register.~~ ✖

TTBR1_EL1 is a <u>mnemonic</u> for Encoding (3, 0, 2, 0, 1)

Access instruction encoding

op0	op1	CRn	CRm	op2

Table D18-2 Instruction encodings for non-debug System register access

op0	op1	CRn	CRm	op2	Access	Mnemonic	Register
11	000	0000	0000	000	RO	MIDR_EL1	MIDR_EL1
11	000	0000	0000	000	RO	MIDR_EL1	VPIDR_EL2
11	000	0000	0000	000	RO	VPIDR_EL2	MIDR_EL1

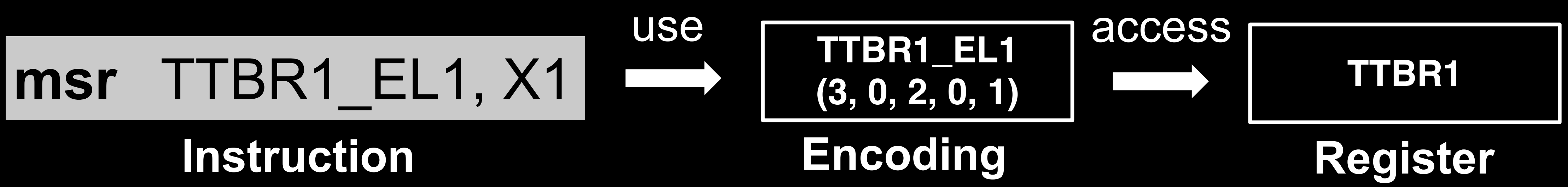
```
MSR TTBR1_EL1, <Xt>
```

op0	op1	CRn	CRm	op2
0b11	0b000	0b0010	0b0000	0b001

Task 1. Apple-spec PAC system register identification

TTBR1_EL1 is a mnemonic for Encoding (3, 0, 2, 0, 1)

msr instruction use Encoding (3, 0, 2, 0, 1) to access Register

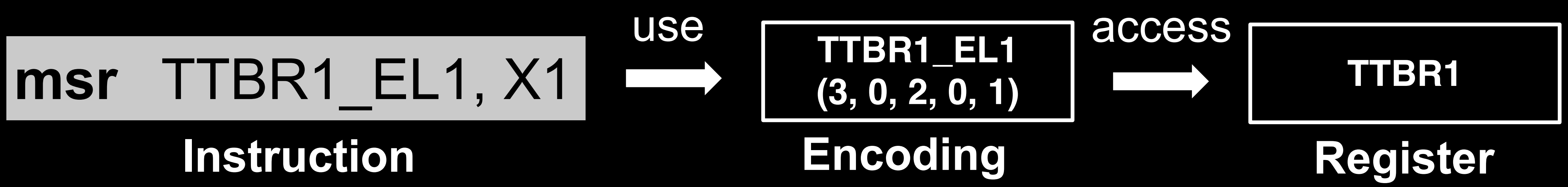


Task 1. Apple-spec PAC system register identification

TTBR1_EL1 is a mnemonic for Encoding (3, 0, 2, 0, 1)

msr instruction use Encoding (3, 0, 2, 0, 1) to access Register

! Encoding and Register are not 1:1 mapping



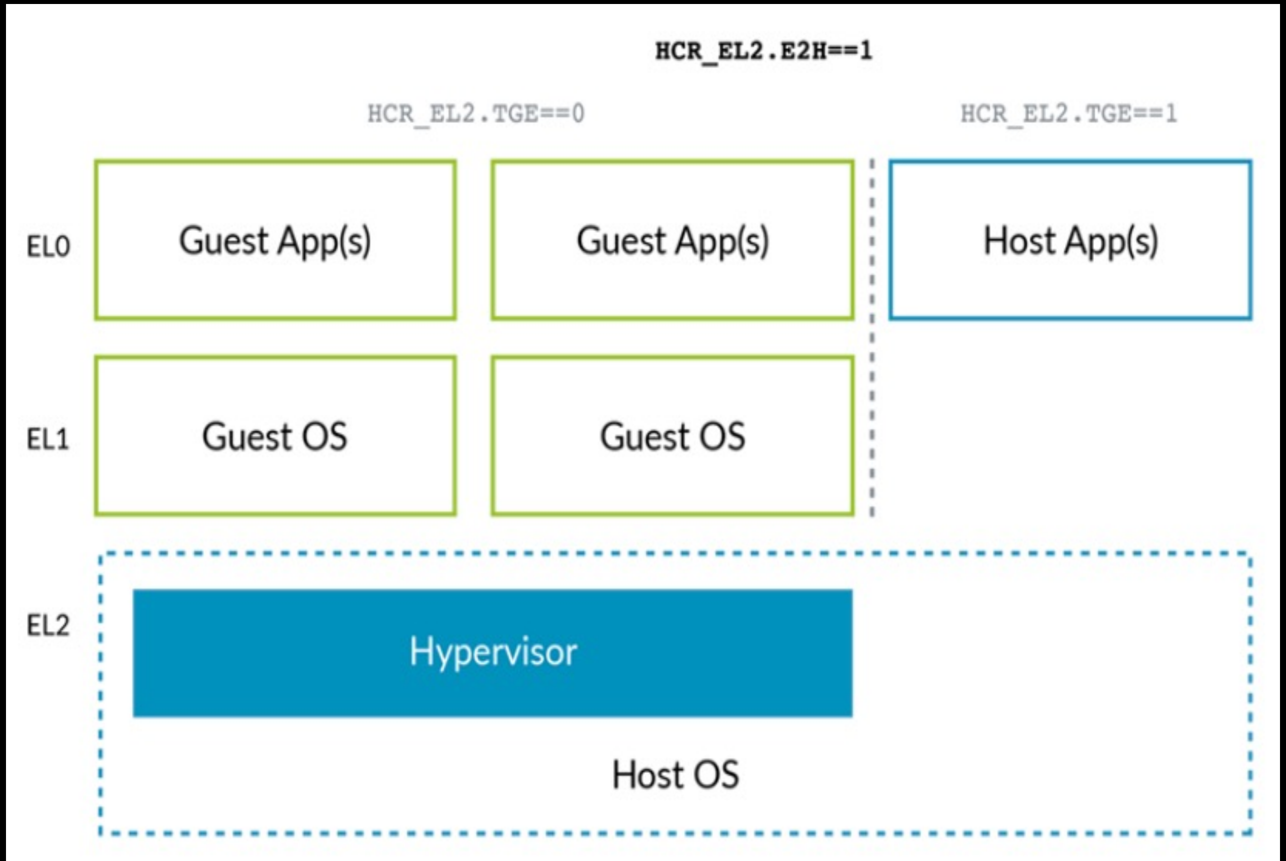
Task 1. Apple-spec PAC system register identification

Virtualization Host Extension (VHE)

- A set of hardware supports for running OS on EL1 and EL2 without software modification

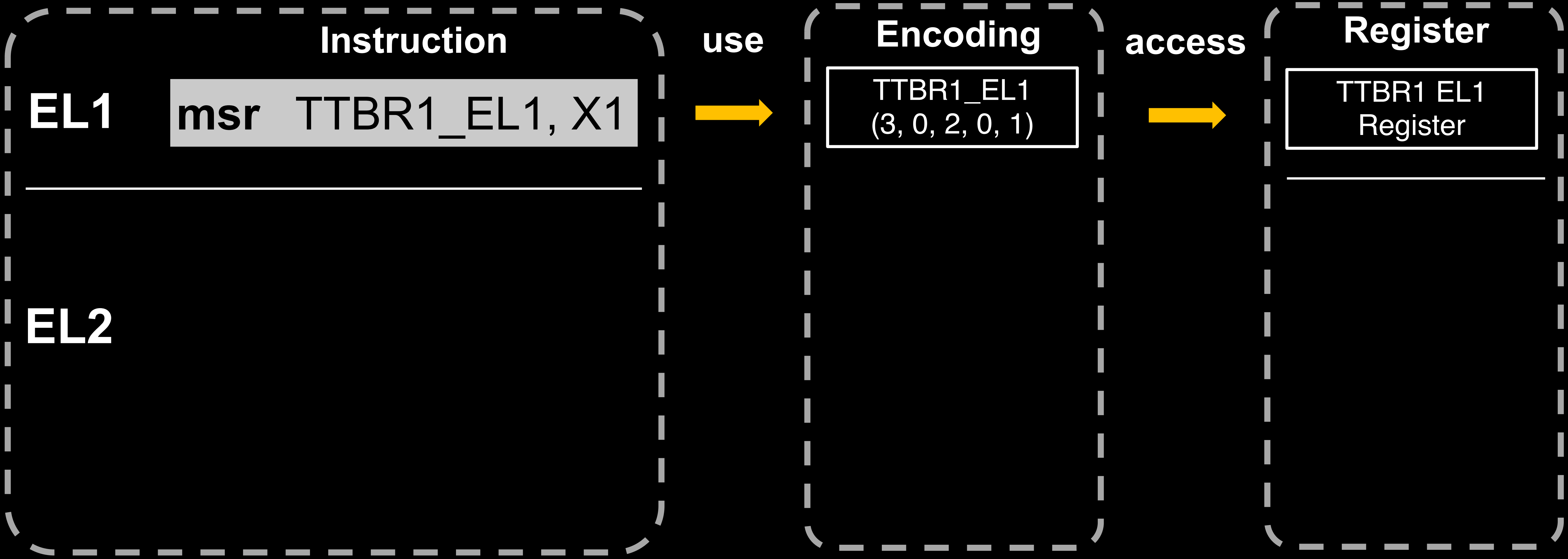
- Hardwired on Apple M1

- Includes **System Register Redirection**



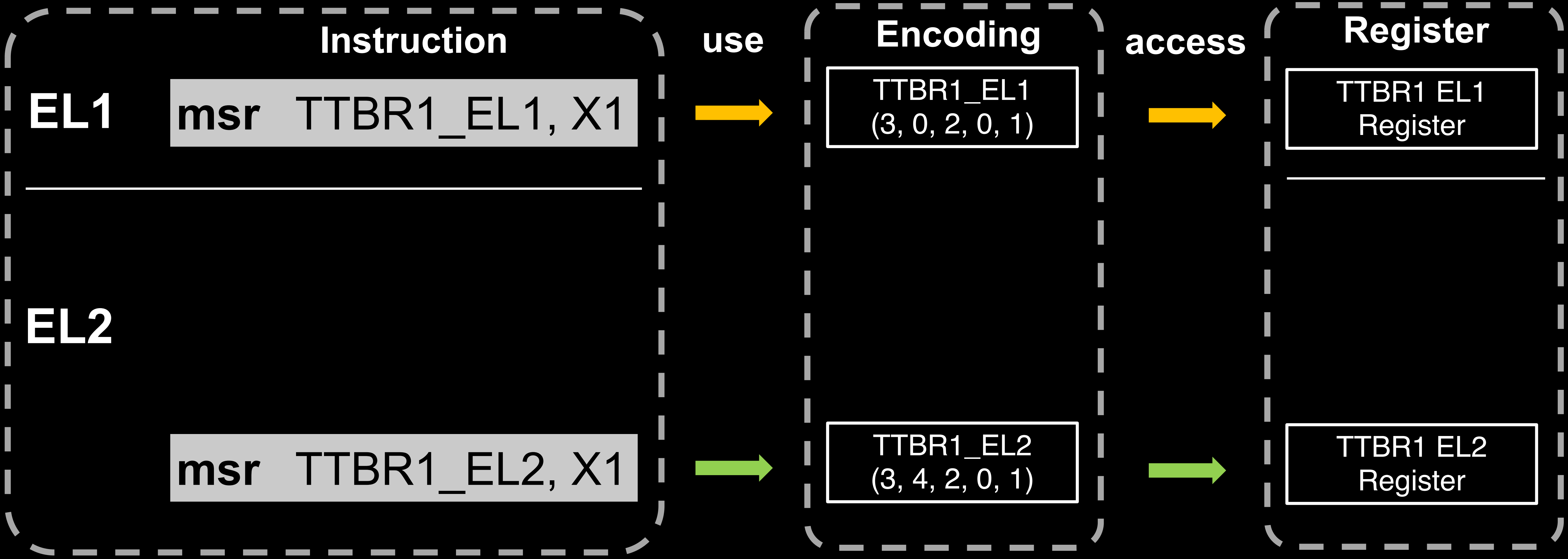
Task 1. Apple-spec PAC system register identification

System Register Redirection



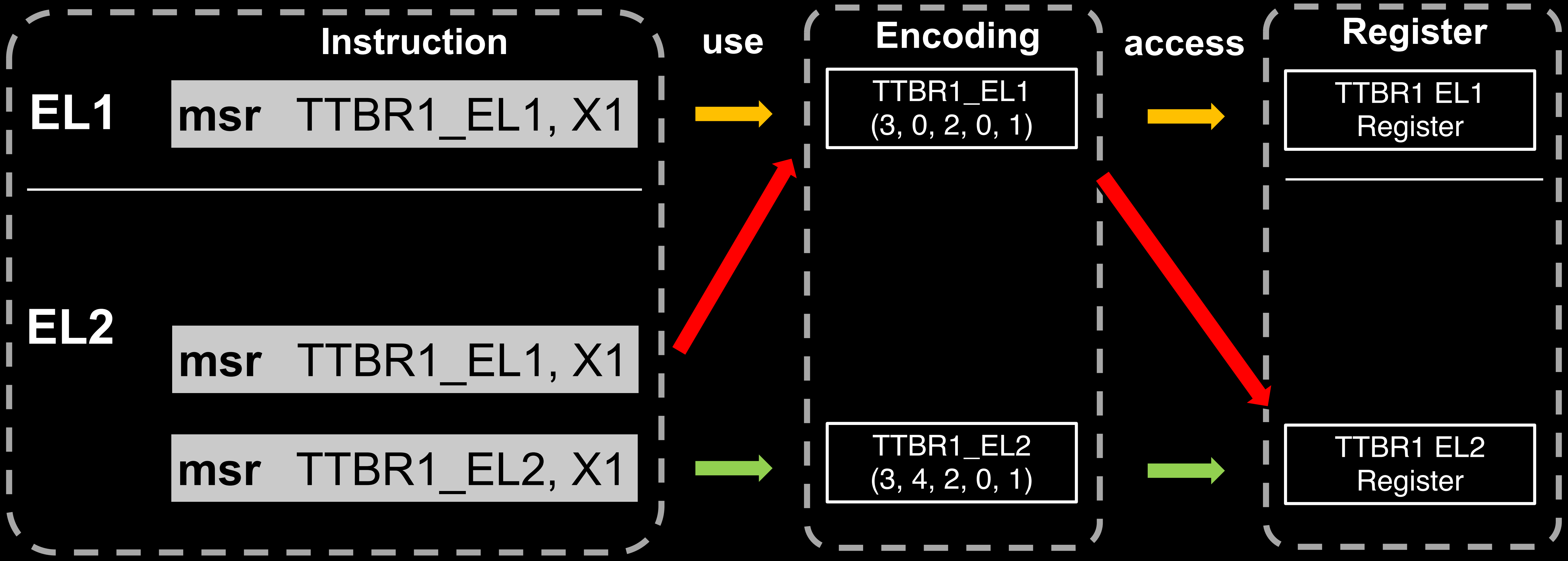
Task 1. Apple-spec PAC system register identification

System Register Redirection



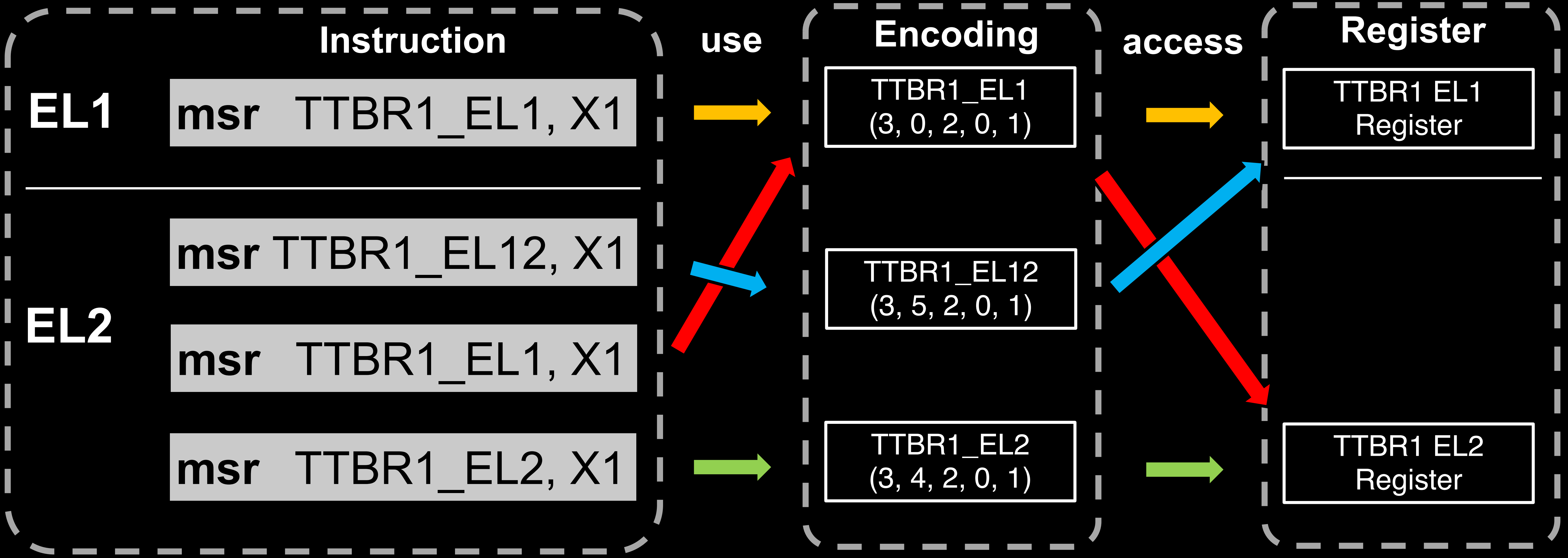
Task 1. Apple-spec PAC system register identification

System Register Redirection



Task 1. Apple-spec PAC system register identification

System Register Redirection



Task 1. Apple-spec PAC system register identification

System Register Redirection

- Bank sysreg on Both EL1 and EL2
- Redirect the Access using EL1 encoding on EL2
- Add a EL12 encoding for accessing EL1 register on EL2

* We term **EL12/EL2 encoding** as **alias encodings**

Task 1. Apple-spec PAC system register identification

Back to Apple-spec Sysreg

Apple introduced a lot of:

New Encodings → (to access) → **New Registers** → (to control) → **New Features**

Task 1. Apple-spec PAC system register identification

Back to Apple-spec Sysreg

Apple introduced a lot of:

New Encodings → to access → **New Registers** → to control → **New Features**

However, Apple doesn't disclose information about them

```
MSR    #6, c15, c14, #4, X8
LDR    X8, [X20,#0x4098]
MSR    #6, c15, c14, #5, X8
```

← **Undisclosed encoding (3, 6, 15, 14, 4)**

The CRn field of Apple-spec Encoding is 15

Task 1. Apple-spec PAC system register identification

```
MSR    #6, c15, c14, #4, X8
LDR    X8, [X20,#0x4098]
MSR    #6, c15, c14, #5, X8
```

Undisclosed encoding (3, 6, 15, 14, 4)

1. How to identify encoding/register of interest?

2. How to understand these encodings/registers?

Task 1. Apple-spec PAC system register identification

1. How to identify encoding/register of interest?

Existing work. (AsahiLinux)

- https://github.com/AsahiLinux/m1n1/blob/main/tools/apple_regs.json


m1n1 / tools / apple_regs.json

Code Blame 334 lines (334 loc) · 44.5 KB

```
{"index": 0, "name": "SPRR_PPERM_EL20_SILLY_THING", "fullname": "SPRR Permission Configuration Register (EL20, useless)",   "enc":
{"index": 0, "name": "SPRR_UPERM_EL02",             "fullname": "SPRR User Permission Configuration Register (EL02)",       "enc":
{"index": 0, "name": "SPRR_UMPRR_EL2",              "fullname": "SPRR User MPRR (EL2)",                                     "enc":
{"index": 0, "name": "SPRR_UPERM_SH1_EL2",          "fullname": "SPRR User Permission SH1 (EL2)",                           "enc":
{"index": 0, "name": "SPRR_UPERM_SH2_EL2",          "fullname": "SPRR User Permission SH2 (EL2)",                           "enc":
{"index": 0, "name": "SPRR_UPERM_SH3_EL2",          "fullname": "SPRR User Permission SH3 (EL2)",                           "enc":
{"index": 0, "name": "SPRR_UMPRR_EL12",             "fullname": "SPRR User MPRR (EL12)",                                    "enc":
{"index": 0, "name": "SPRR_UPERM_SH1_EL12",         "fullname": "SPRR User Permission SH1 (EL12)",                          "enc":
{"index": 0, "name": "SPRR_UPERM_SH2_EL12",         "fullname": "SPRR User Permission SH2 (EL12)",                          "enc":
{"index": 0, "name": "SPRR_UPERM_SH3_EL12",         "fullname": "SPRR User Permission SH3 (EL12)",                          "enc":
{"index": 0, "name": "CTRR_A_LWR_EL12",             "fullname": "CTRR A Lower Address (EL12)",                              "enc":
{"index": 0, "name": "CTRR_A_UPR_EL12",             "fullname": "CTRR A Upper Address (EL12)",                              "enc":
{"index": 0, "name": "CTRR_B_LWR_EL12",             "fullname": "CTRR B Lower Address (EL12)",                              "enc":
{"index": 0, "name": "CTRR_B_UPR_EL12",             "fullname": "CTRR B Upper Address (EL12)",                              "enc":
{"index": 0, "name": "CTRR_CTL_EL12",               "fullname": "CTRR Control (EL12)",                                      "enc":
```

Task 1. Apple-spec PAC system register identification

1. How to identify/document encoding/register of interest?

Tip 1. String Data/ Function/ Known Sysreg in Binary

osfmk > arm64 > C platform_tests.c

```c
kern_return_t
arm64_ropjop_test()
{
    T_LOG("Testing ROP/JOP");

    /* how is ROP/JOP configured */
    boolean_t config_rop_enabled = TRUE;
    boolean_t config_jop_enabled = TRUE;


    if (config_jop_enabled) {
        /* jop key */
        uint64_t apiakey_hi = __builtin_arm_rsr64("APIAKEYHI_EL1");
        uint64_t apiakey_lo = __builtin_arm_rsr64("APIAKEYLO_EL1");

        T_EXPECT(apiakey_hi != 0 && apiakey_lo != 0, NULL);
    }
```

XNU kernel open-source code

```
; arm64_ropjop_test
...
mrs     X8, #6, c15, c12, #4 ; APSTS_EL1
...
and     W8, W8, #1
adrp    X24, #_ktest_temp1@PAGE
str     W8, [X24,#_ktest_temp1@PAGEOFF]
adrl    X0, aApsts1ull0 ; "apsts & (1ULL << 0)"
bl      _ktest_set_current_expr ; if test fails,
    ↪ panic will happen and the message above will
    ↪ be printed
...
```

XNU kernel binary

The code related to Apple-spec sysreg can only be viewed in Binary

Task 1. Apple-spec PAC system register identification

1. How to identify/document encoding/register of interest?

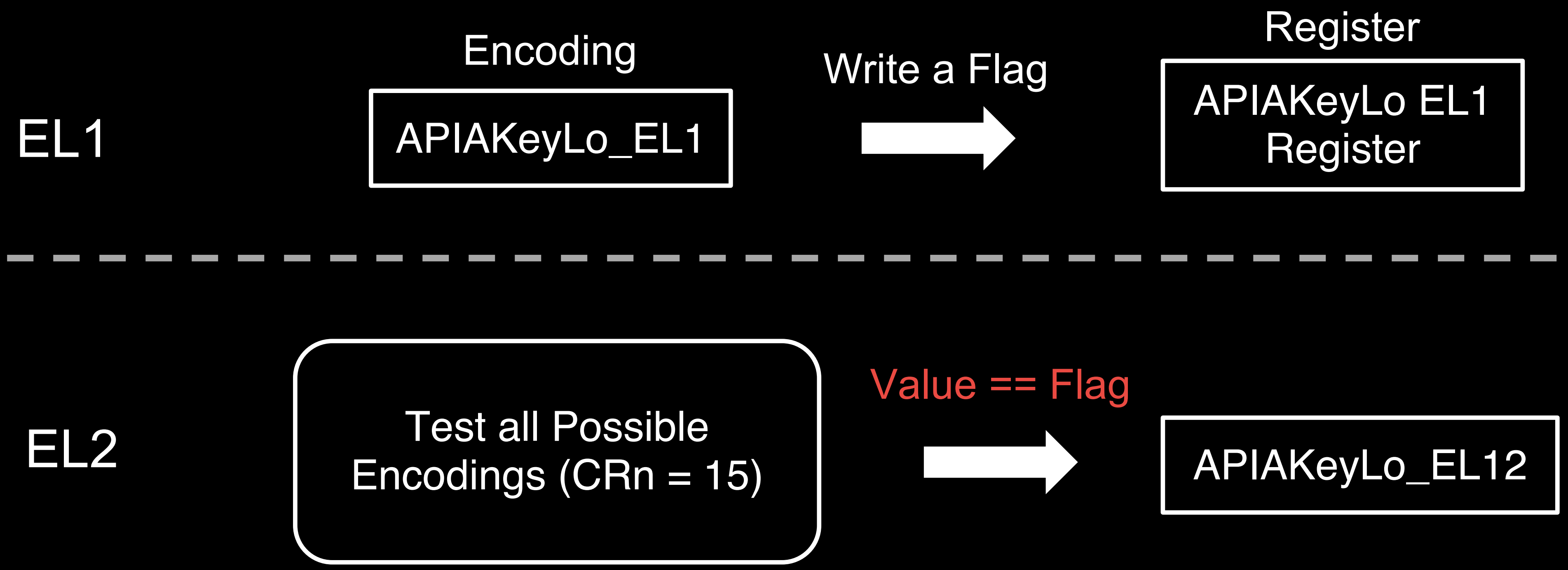
Tip 2. Alias encoding (EL12/EL2)

	Encoding		Register
EL1	APIAKeyLo_EL1	Write a Flag →	APIAKeyLo EL1 Register

Task 1. Apple-spec PAC system register identification

1. How to identify/document encoding/register of interest?

Tip 2. Alias encoding (EL12/EL2)



Not Applicable for all cases (e.g., PAC Key EL2 encoding)

Task 1. Apple-spec PAC system register identification

1. How to identify/document encoding/register of interest?

Tip 3. Identify more encodings based on Alias encoding

```
...
; in the same basic block
ldr         x8, [x20, #0x40a0]
msr         #6, c15, c14, #4, x8 ; VMDIVLo_EL2
ldr         x8, [x20, #0x4098]
msr         #6, c15, c14, #5, x8 ; VMDIVHi_EL2
ldr         x8, [x20, #0x40a8]
msr         #6, c15, c14, #7, x8 ; APSTS_EL12
...
```

There's no info in Binary for VMDIVLo (3, 6, 15, 14, 4), we mark it as PAC-related based on identified alias encoding and tests

Task 1. Apple-spec PAC system register identification

2. How to understand the usage of these encoding/register?

Tip 1. Manually analysis

Some Sysregs are set up with hard-coded value

Task 1. Apple-spec PAC system register identification

2. How to understand the usage of these encoding/register?

Tip 2. Dynamic analysis – Sniff Sysregs

Based on m1n1 hypervisor

- https://github.com/AsahiLinux/m1n1/tree/main

We implement a hypervisor-based XNU kernel debugger

- Active kernel debugging
- Unlimited number of breakpoints

We plan to open-source it this year. (co-work with Jiaxun Zhu @svnswords)

Task 1. Apple-spec PAC system register identification

2. How to understand the usage of these encoding/register?

Tip 3. Run your tests on EL1 first

Most Apple-spec feature are deployed on both EL1 and EL2

- Trap into EL2 to observe EL1 things with higher privilege

Task 1. Apple-spec PAC system register identification

Almost all easy(general) cases are done

However, there are still lots of undocumented encodings

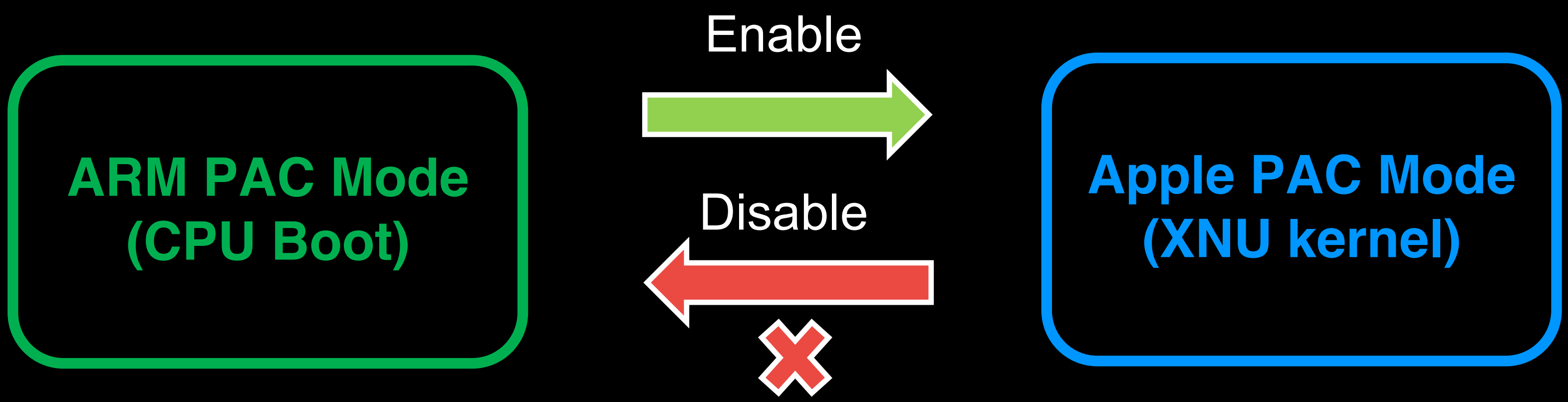
- Not used in the XNU kernel

We need your help for more tests to document them

Task 2. Apple-spec PAC Key Protection Bypassing

Task 2. Apple-spec PAC Key Protection Bypassing

Two PAC modes on Apple M1



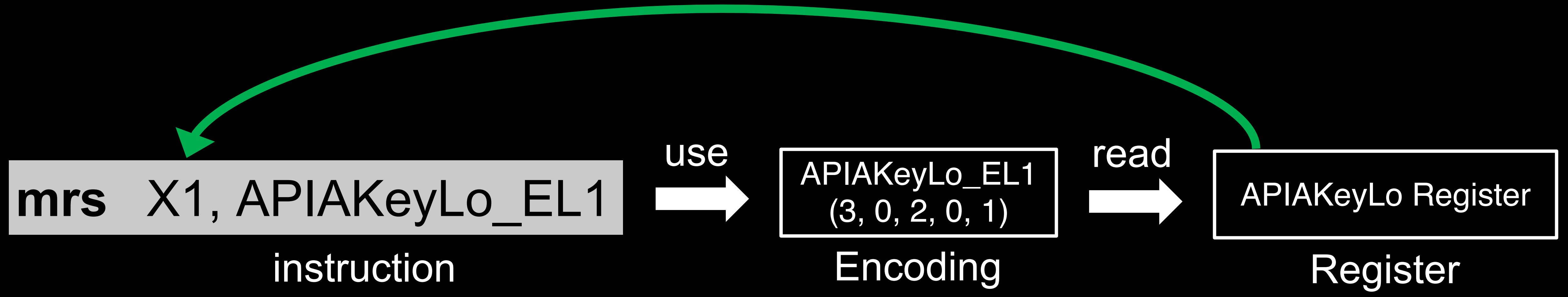
Our Target: Profile the PAC instruction behavior after enabling Apple PAC Mode

Task 2. Apple-spec PAC Key Protection Bypassing

Apple-spec PAC Key Protection

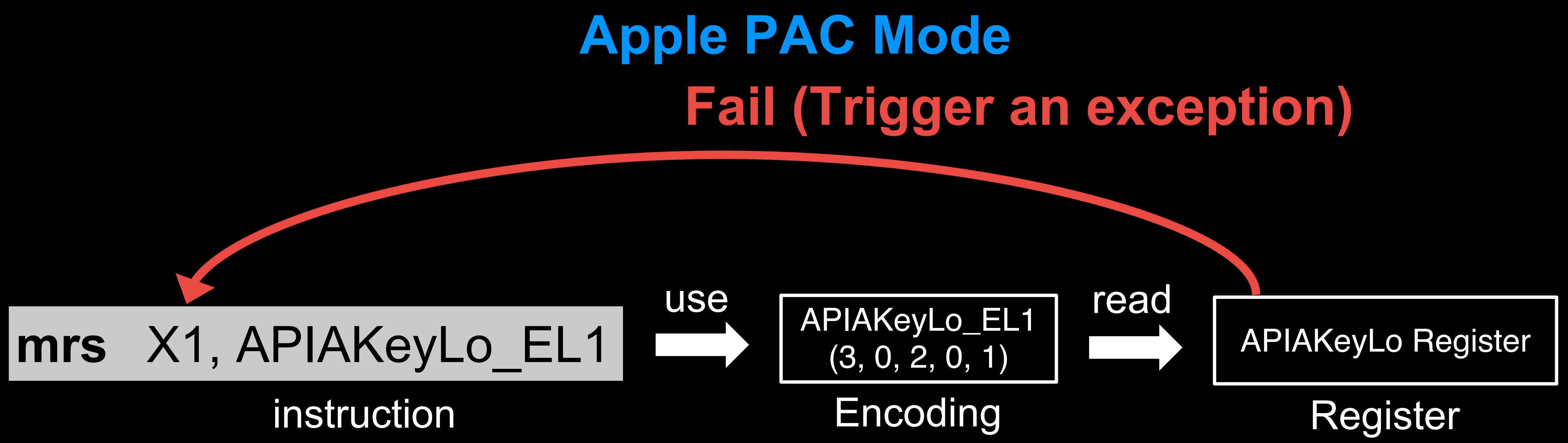
ARM PAC Mode

Success



Task 2. Apple-spec PAC Key Protection Bypassing

Apple-spec PAC Key Protection



Task 2. Apple-spec PAC Key Protection Bypassing

Why we need to bypass PAC Key Protection

The inputs we can control:

- Key Value (set)
- Key Selection
- Pointer
- Modifier

```
pacxx   x1, x2
```

```
msr   Key_EL1, x1
```

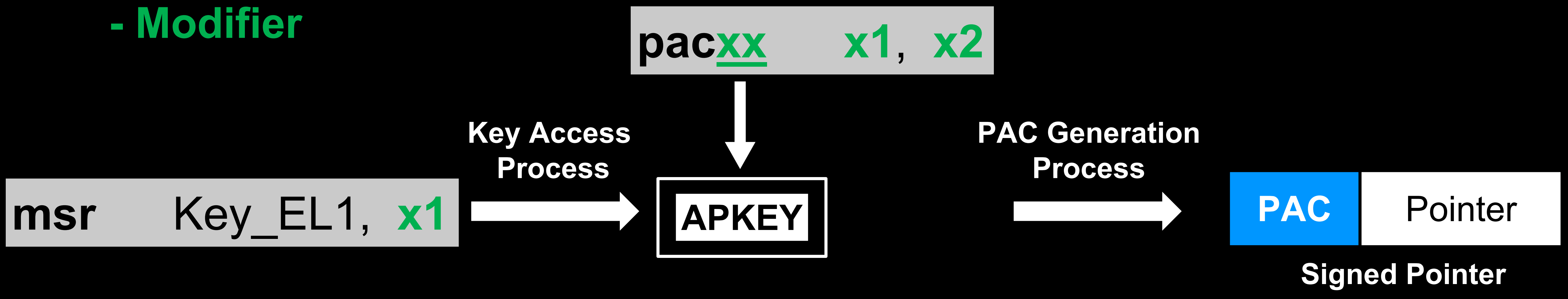
Key Access Process → APKEY → PAC Generation Process

Task 2. Apple-spec PAC Key Protection Bypassing

Why we need to bypass PAC Key Protection

The inputs we can control:
- Key Value (set)
- Key Selection
- Pointer
- Modifier

The output we can read:
- PAC result



Task 2. Apple-spec PAC Key Protection Bypassing

Why we need to bypass PAC Key Protection

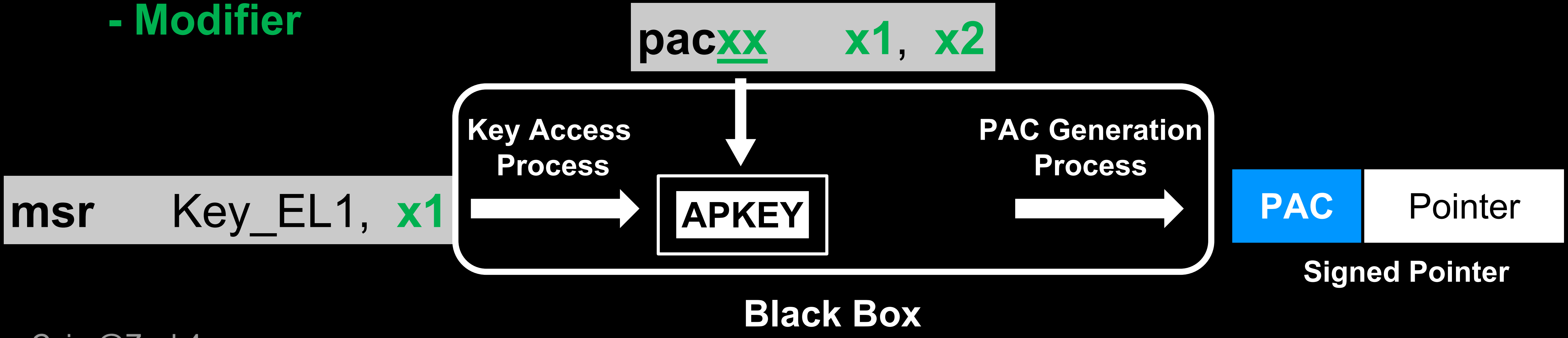
The inputs we can control:

- Key Value (set)
- Key Selection
- Pointer
- Modifier

The output we can read:

- PAC result

We can't determine "Dart Magic" is happened in which process



Task 2. Apple-spec PAC Key Protection Bypassing

Why we need to bypass PAC Key Protection

The inputs we can control:

- Key Value (set)
- Key Selection
- Pointer
- Modifier

The output we can read:

- PAC result

If we can read the key

- APKEY

We **can** determine "Dart Magic" happened in which process

```
pacxx    x1,  x2
```

```
msr    Key_EL1,  x1
```

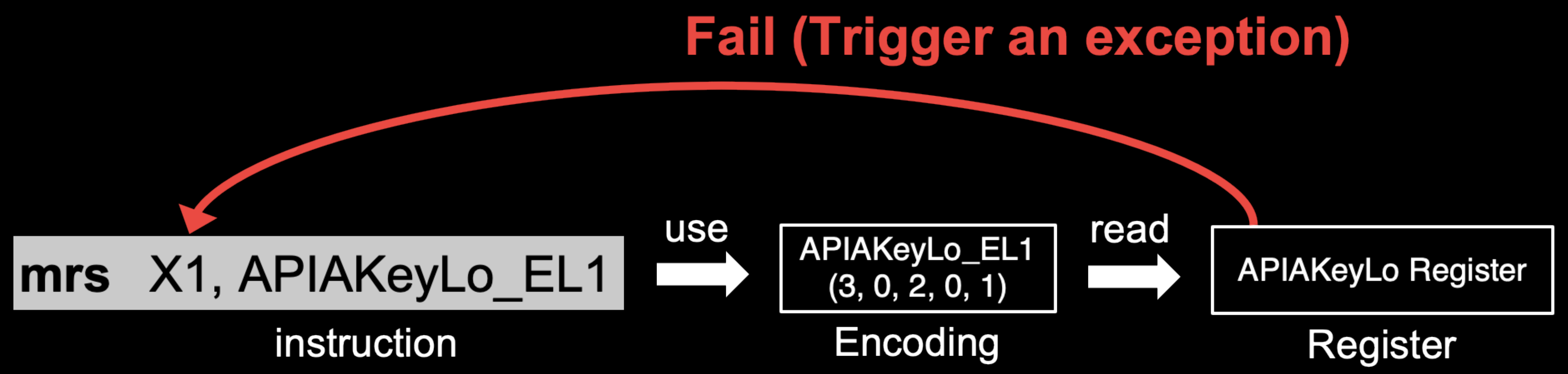
Key Access Process → **APKEY** → **PAC Generation Process** → **PAC** | Pointer

Signed Pointer

Task 2. Apple-spec PAC Key Protection Bypassing

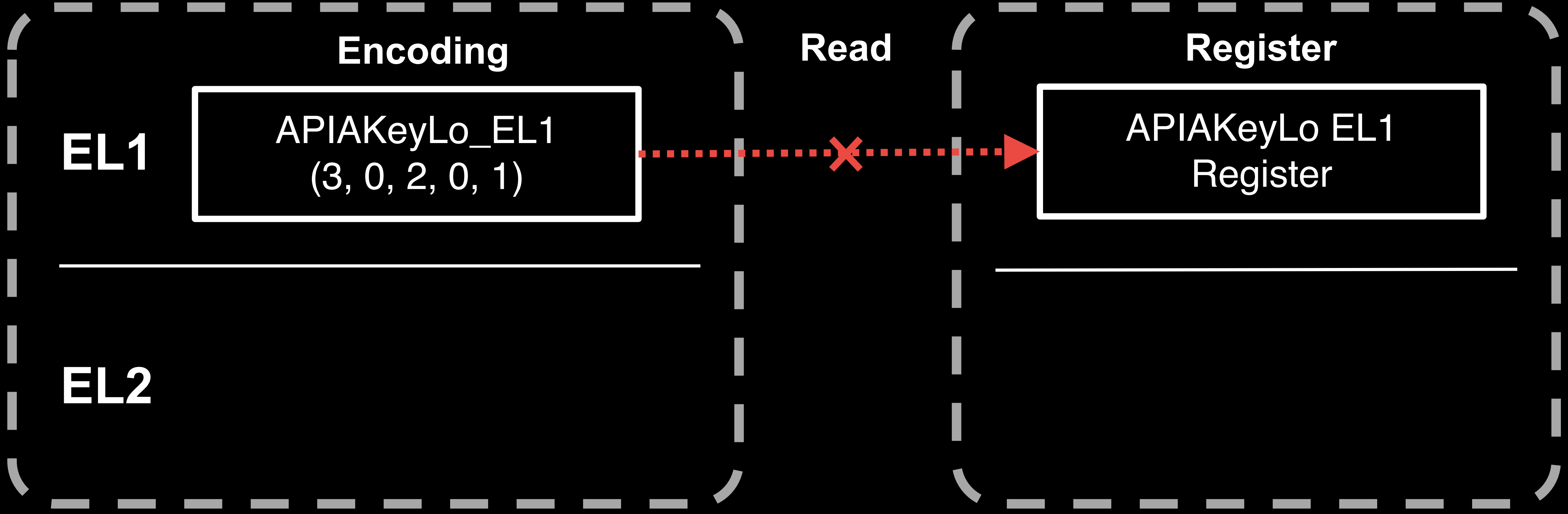
Apple-spec PAC Key Protection

- Deployed on both EL1 and EL2
  - Apple PAC is different on EL1 and EL2
- EL1 Key Protection Bypass
- EL2 Key Protection Bypass



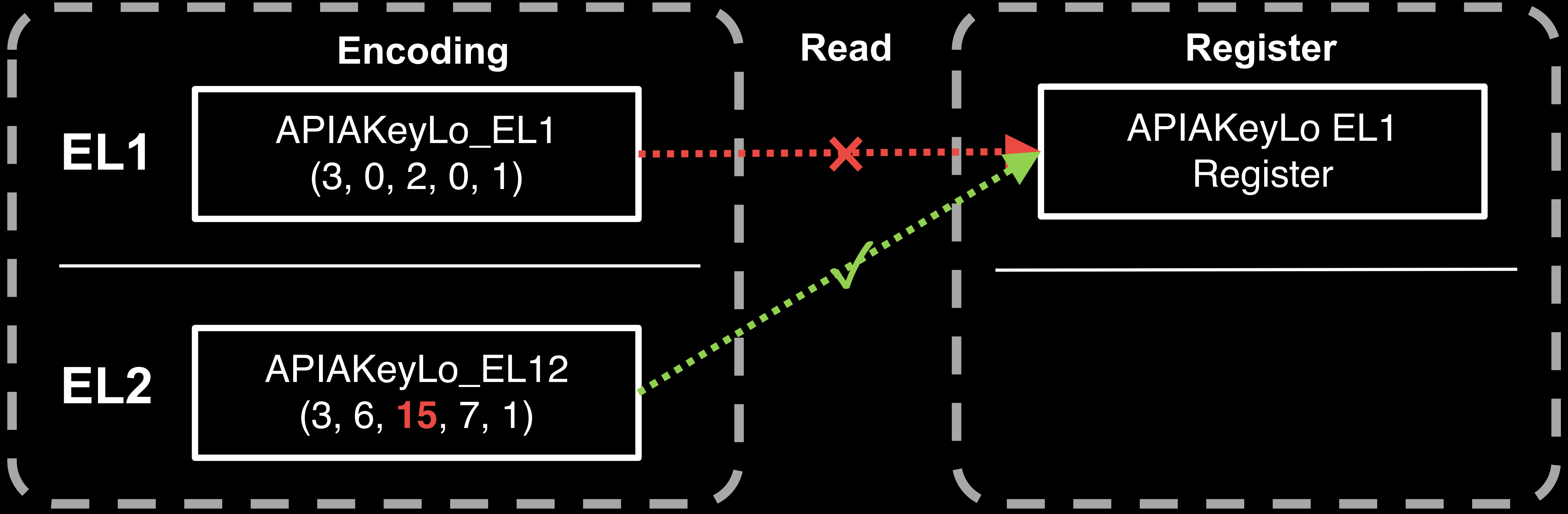
Task 2. Apple-spec PAC Key Protection Bypassing

EL1 Key Protection Bypass



Task 2. Apple-spec PAC Key Protection Bypassing

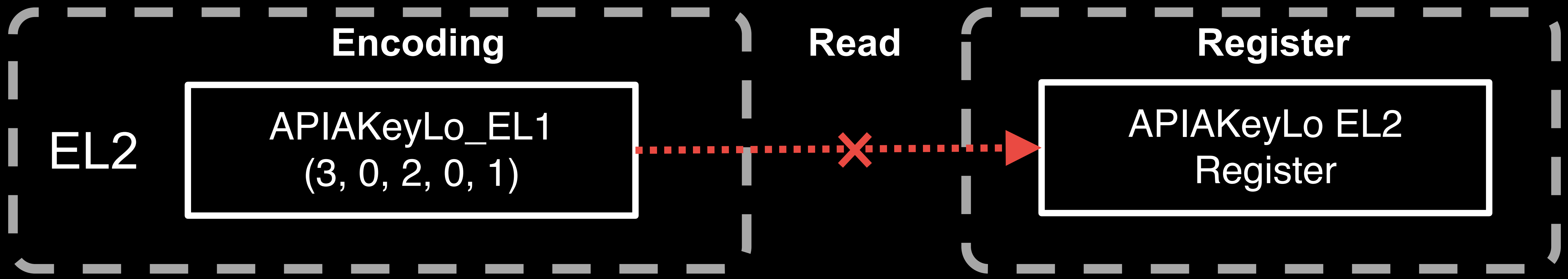
EL1 Key Protection Bypass



Task 2. Apple-spec PAC Key Protection Bypassing

EL2 Key Protection Bypass

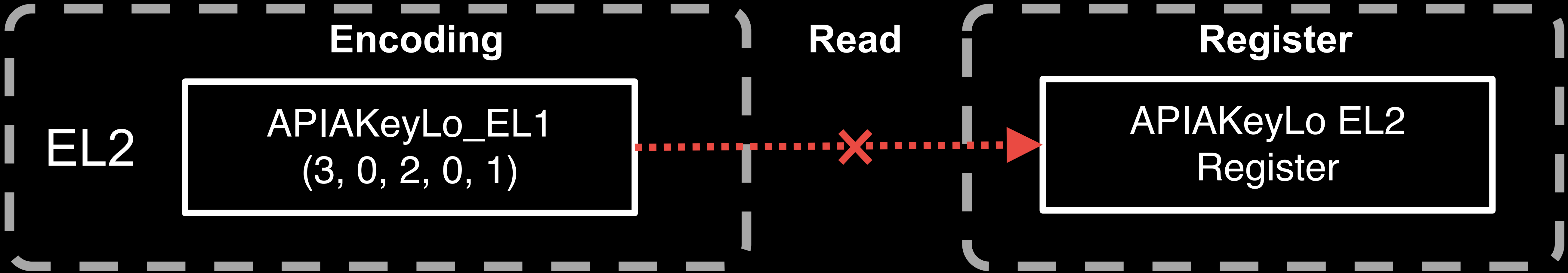
- There is no higher Exception Level (EL3) on Apple M1



Task 2. Apple-spec PAC Key Protection Bypassing

EL2 Key Protection Bypass

- There is no higher Exception Level (EL3) on Apple M1

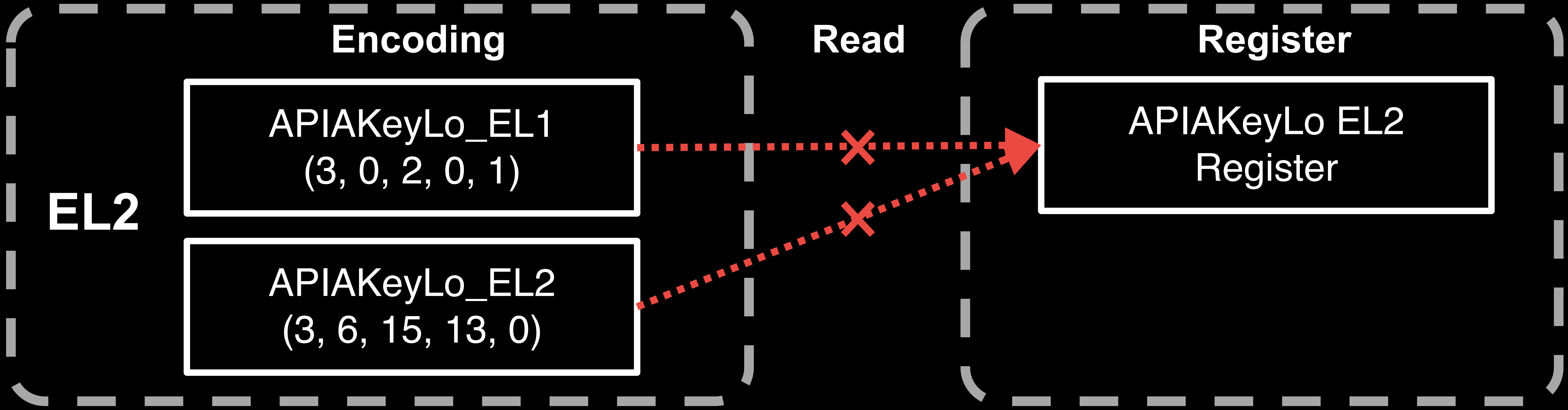


Idea 1: Are there other encodings for accessing the PAC Key?

Task 2. Apple-spec PAC Key Protection Bypassing

EL2 Key Protection Bypass

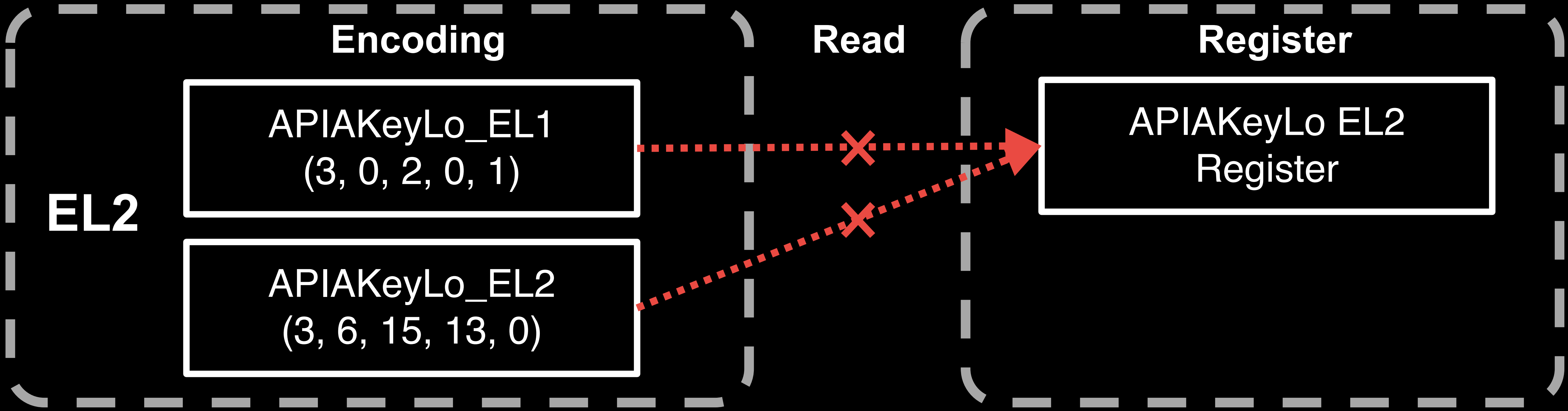
- There is no higher Exception Level (EL3) on Apple M1
- EL2 PAC Key Encoding is also **Non-Readable**



Task 2. Apple-spec PAC Key Protection Bypassing

EL2 Key Protection Bypass

- There is no higher Exception Level (EL3) on Apple M1



Idea 2: Side-channel Attack? ✕

Task 2. Apple-spec PAC Key Protection Bypassing

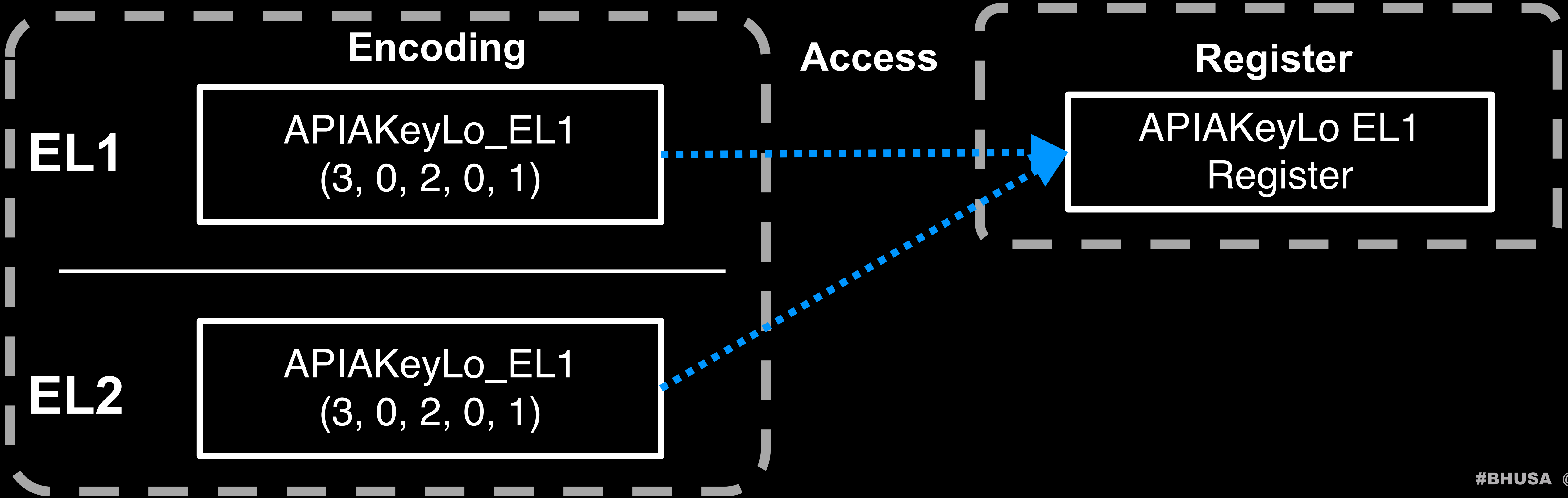
EL2 Key Protection Bypass

A Lot of Tests

Task 2. Apple-spec PAC Key Protection Bypassing

Observation 1

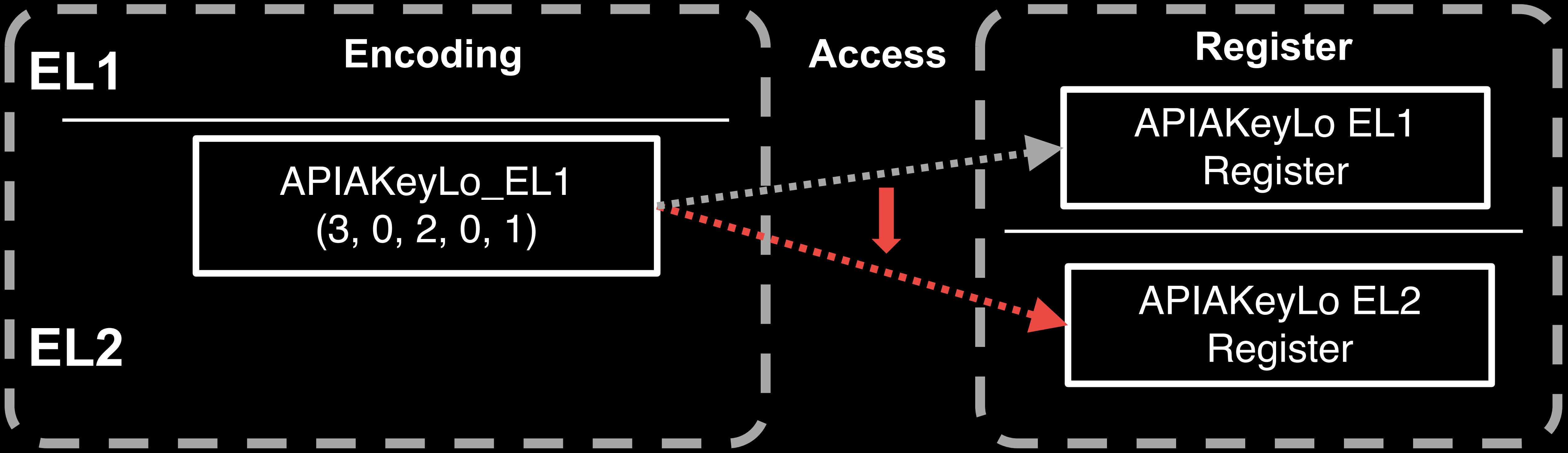
- If Apple PAC mode is disable on EL2
- **Only one set of PAC Keys** are enabled



Task 2. Apple-spec PAC Key Protection Bypassing

Observation 1

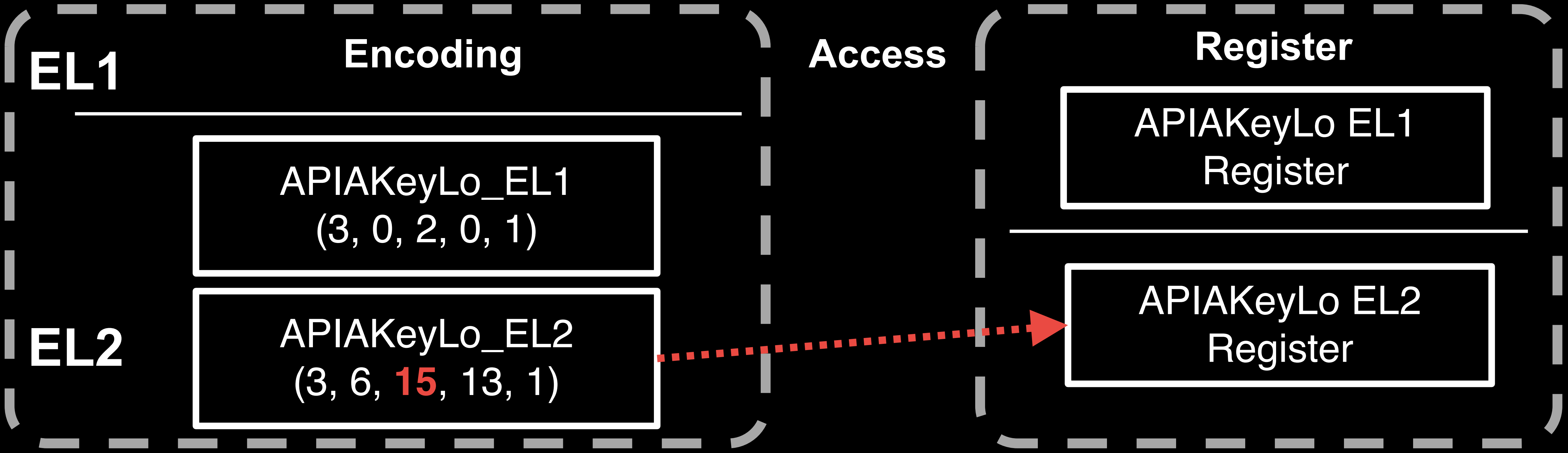
- The access of EL1 Key encoding **changes after Apple PAC is enabled**



Task 2. Apple-spec PAC Key Protection Bypassing

Observation 2

- Enabling Apple PAC won't change the value in EL2 PAC Key Register



Task 2. Apple-spec PAC Key Protection Bypassing

Observation 3

- **PAC calculation is based on** the key value accessed by **EL1 encoding**

Instruction

pacia x1, x2

use →

Encoding

APIAKeyLo_EL1 (3, 0, 2, 0, 1)

APIAKeyHi_EL1 (3, 0, 2, 0, 2)

access →

Register

APIAKeyLo EL1 Register

APIAKeyHi EL1 Register

Task 2. Apple-spec PAC Key Protection Bypassing

Why we need to bypass PAC Key Protection

The inputs we can control:

- Key Value (set)
- Key Selection
- Pointer
- Modifier

The output we can read:

- PAC result

If we can read the key

- APKEY

We **can** determine "Dart Magic" happened in which process

pac**xx** **x1**, **x2**

msr Key_EL1, **x1**

Key Access Process → APKEY → PAC Generation Process → PAC | Pointer

Signed Pointer

What we need: Determine the PAC Key value used for PAC Calculation when Apple PAC is enabled

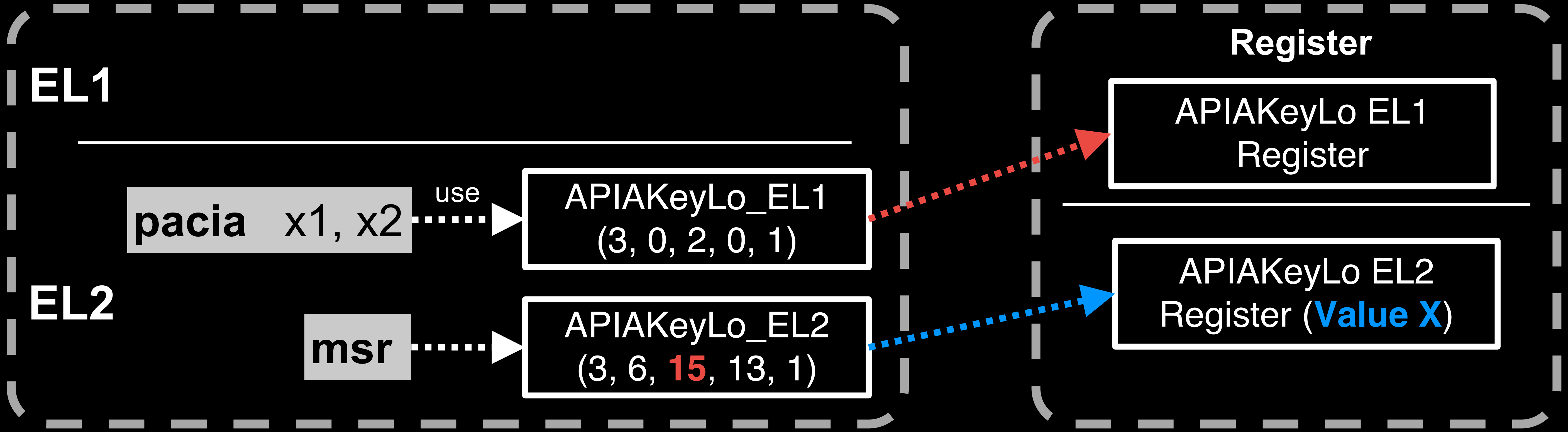
Task 2. Apple-spec PAC Key Protection Bypassing

EL2 Key Protection Bypass

Idea: Preset the PAC Keys before Apple PAC is enabled

Task 2. Apple-spec PAC Key Protection Bypassing

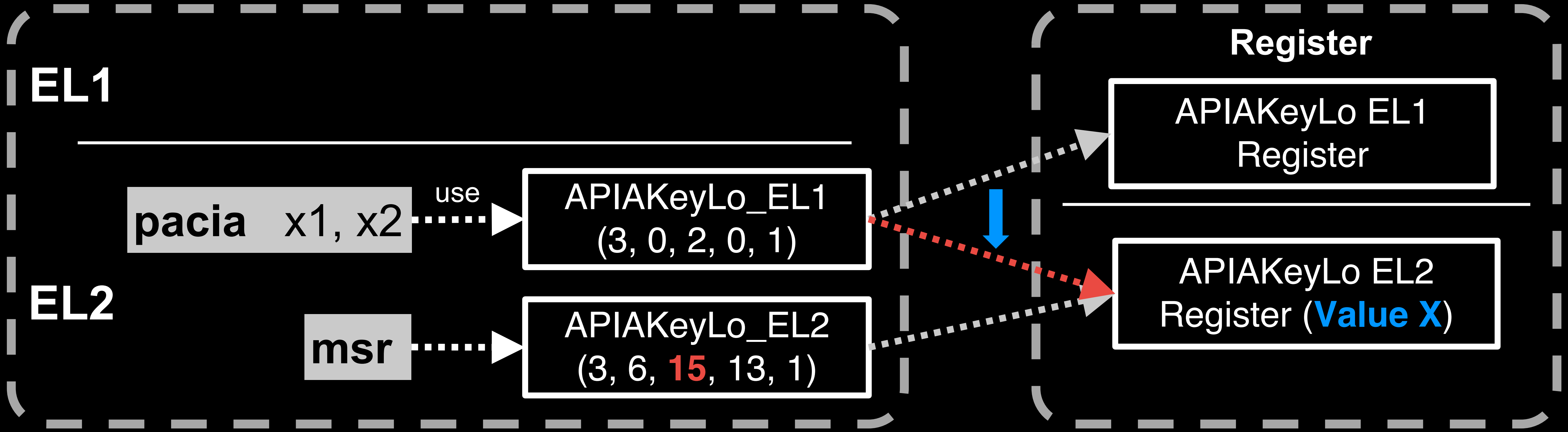
EL2 Key Protection Bypass



Step 1. Set up the EL2 PAC Key using EL2 Encoding with Value X

Task 2. Apple-spec PAC Key Protection Bypassing

EL2 Key Protection Bypass



Step 2. Enable the Apple PAC, the pac inst will calculate PAC based on Value X

Reverse Engineering

Change CPU States and See what happens

Set System Register

Run Instructions

Step 1

Step 2

Our Findings

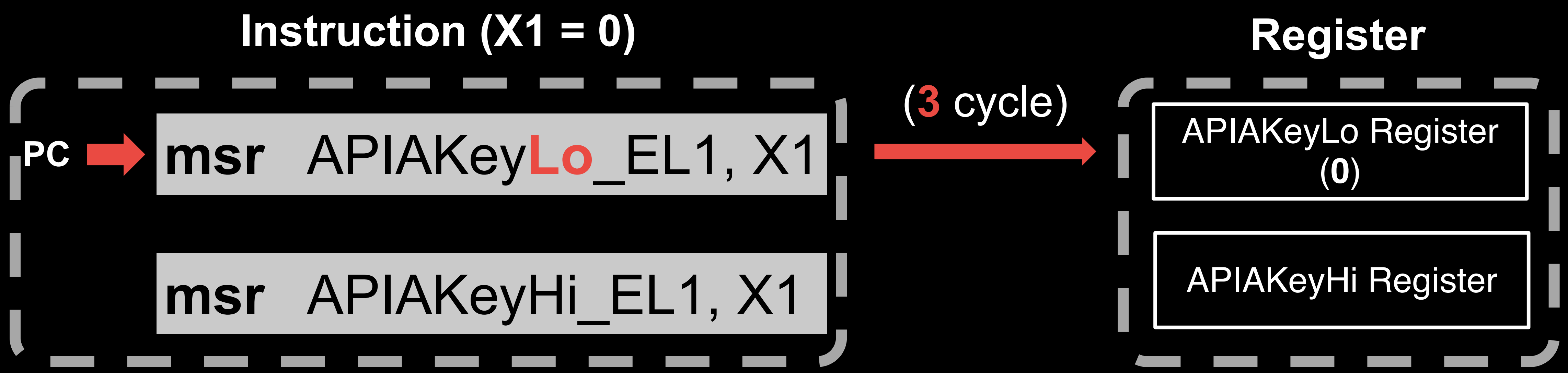
Apple's Customization on PAC Hardware

Finding Overview

- Register
  - APCTL_EL1 (Apple-spec PAC Control Register)
  - EXTRAKEY_EL1 (128-bit User-Kernel Diversifier)
  - VMDIV_EL2 (128-bit Per-VM Diverisifer)
- Instruction
  - Key Access
  - pac/aut

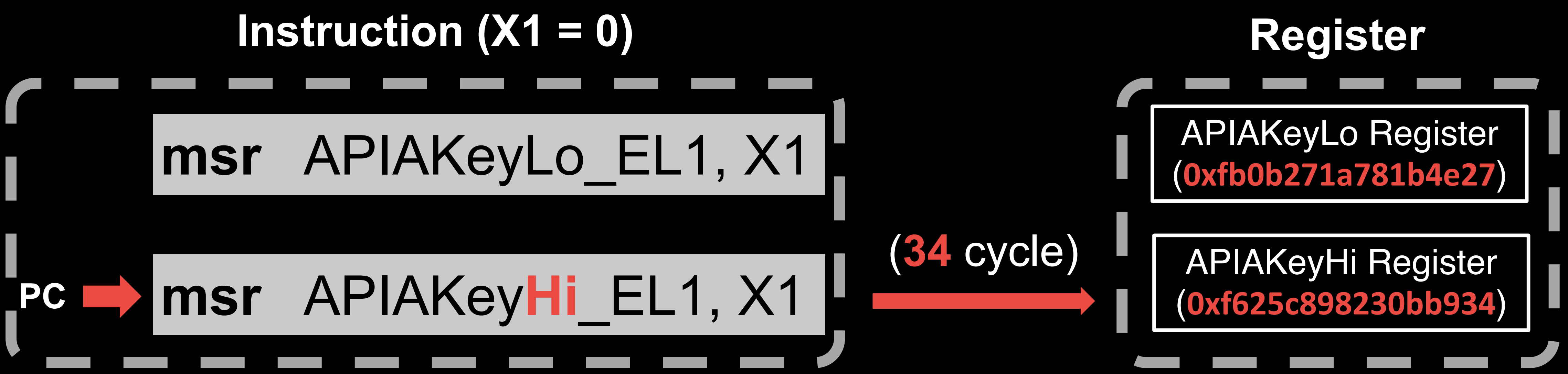
Apple's Customization on PAC Hardware

Key Access



Apple’s Customization on PAC Hardware

Key Access

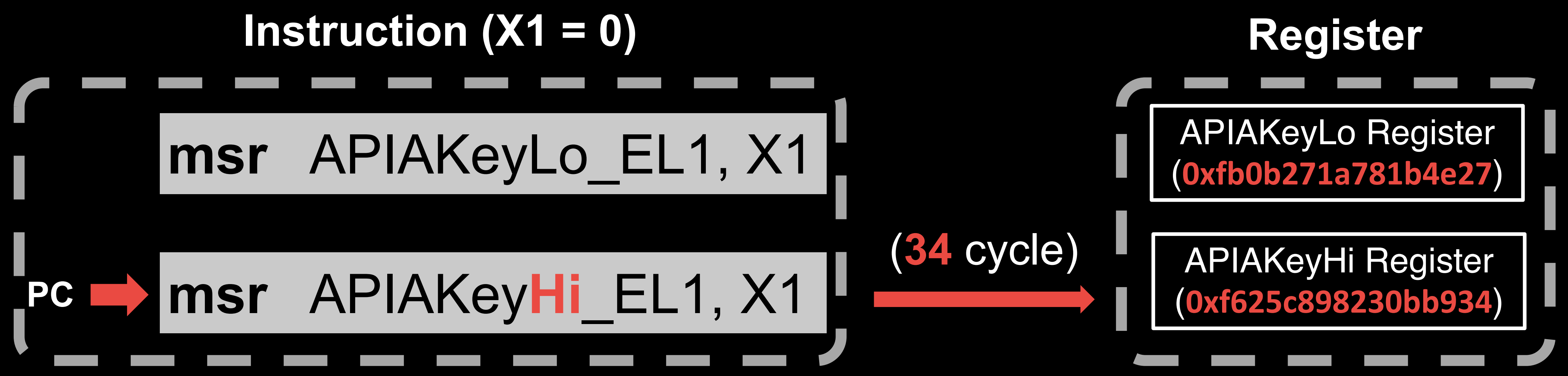


black hat USA 2023

Apple's Customization on PAC Hardware

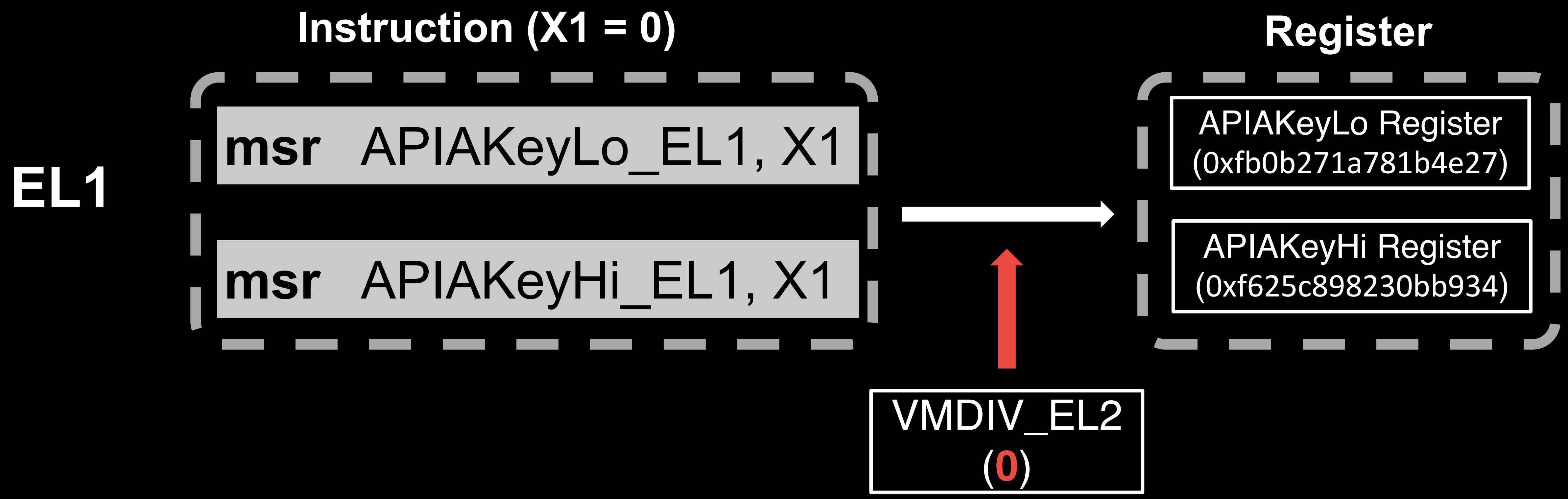
Key Access

Set up the **higher 64 bits of PAC Key** will trigger a **Key Transformation**



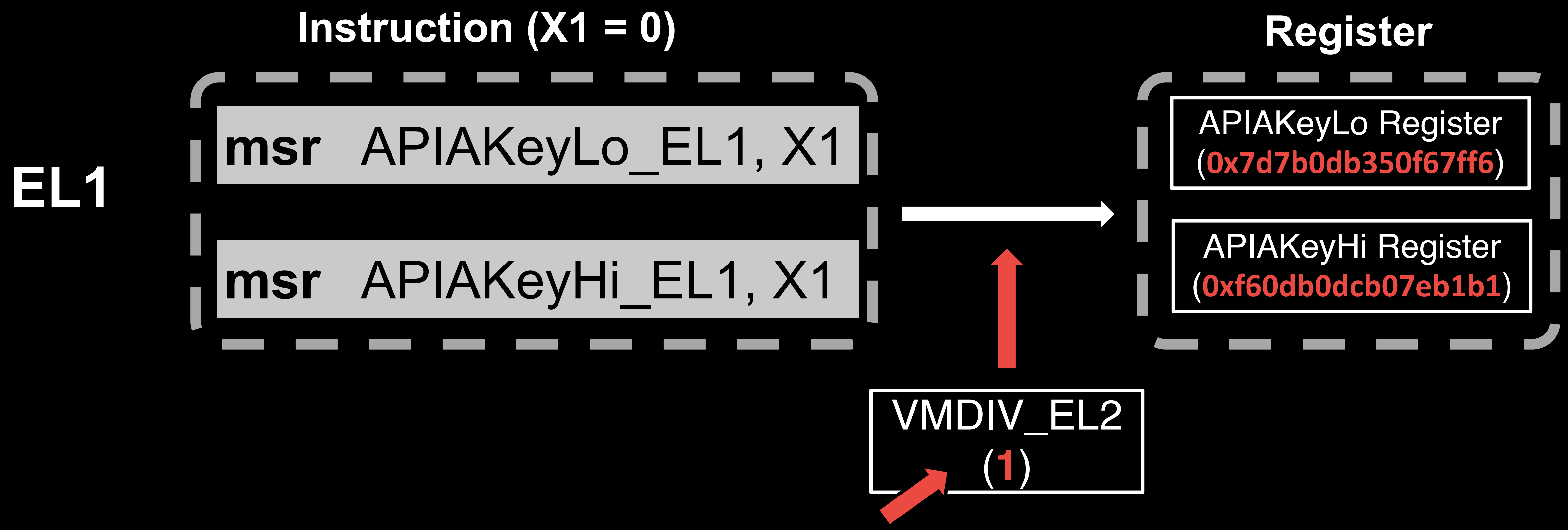
Apple's Customization on PAC Hardware

Key Access



Apple's Customization on PAC Hardware

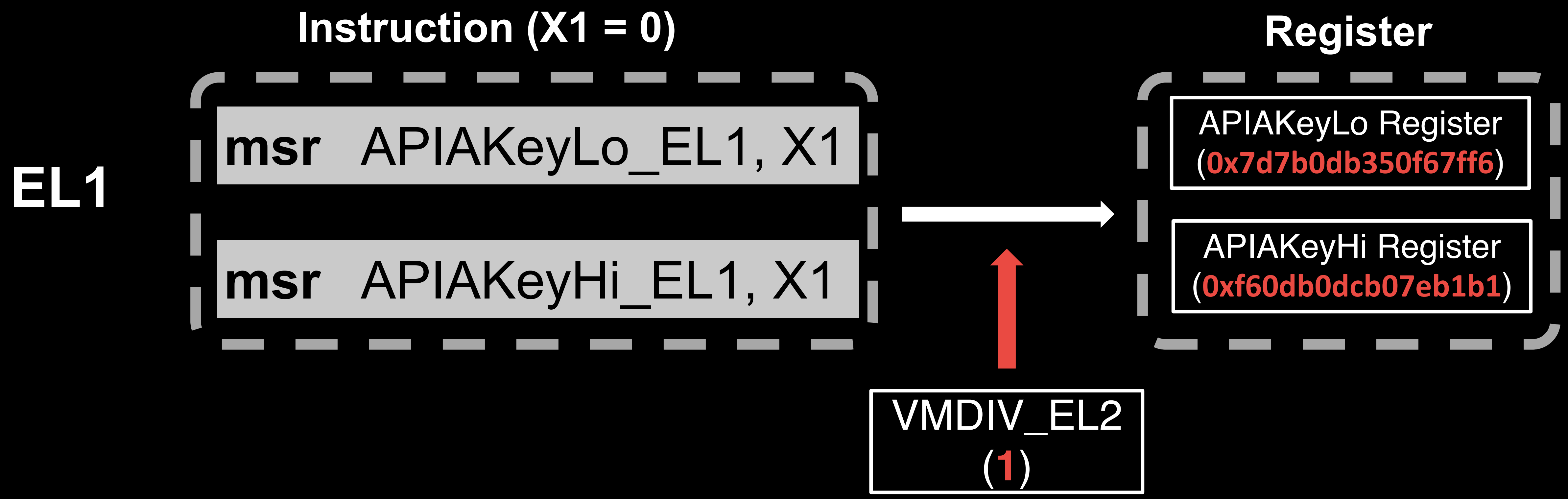
Key Access



Set up the VMDIV_EL2 with different value and trigger the EL1 Key Transformation

Apple’s Customization on PAC Hardware

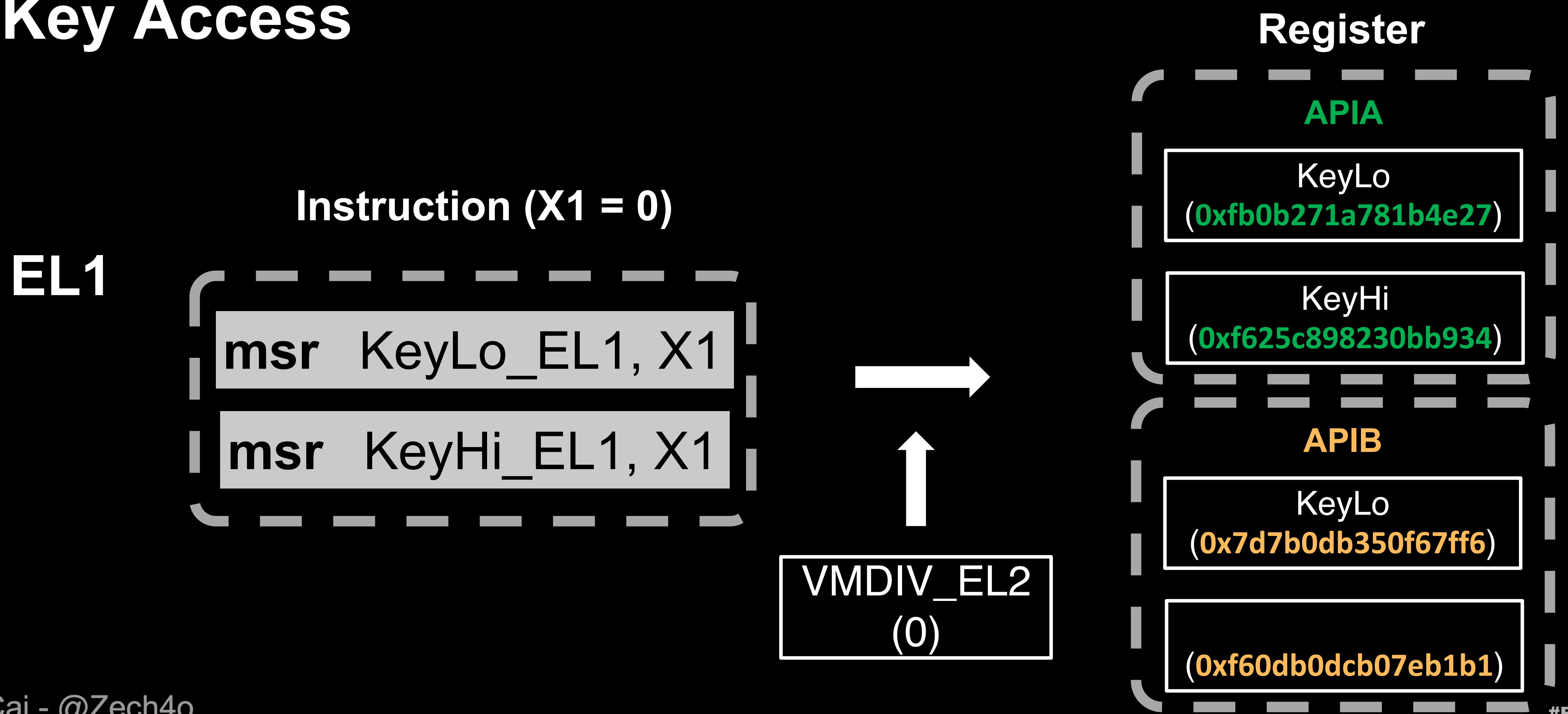
Key Access



VMDIV_EL2 is one of inputs for **EL1 Key Transformation**

Apple’s Customization on PAC Hardware

Key Access



Apple's Customization on PAC Hardware

Key Access

How Apple differentiate Key Transformation for different Key?

Apple's Customization on PAC Hardware

Key Access

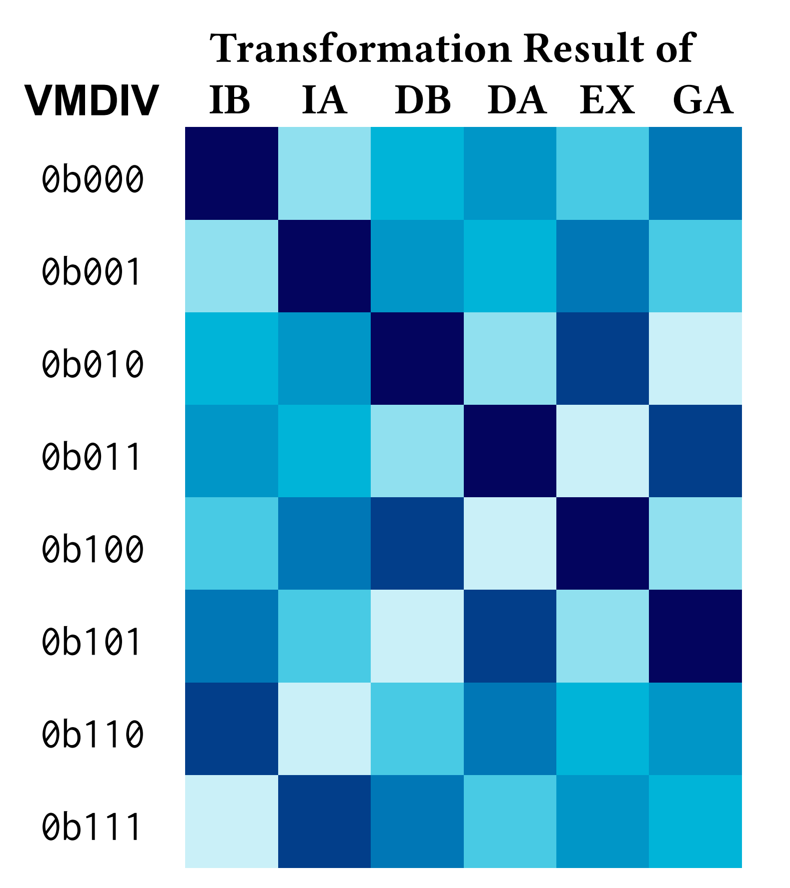
I set the VMDIV from 0b000 to 0b111

VMDIV	Transformation Result of					
	IB	IA	DB	DA	EX	GA
0b000	0x7d7b0db350f67ff6 0xf60db0dcb07eb1b1	0xfb0b271a781b4e27 0xf625c898230bb934	0xe2ee9eaaa4ec5479 0x3cd6dc8228c5488d	0x3e2b1b189fbc10b4 0xe97d268ae2681267	0xb455818159de0818 0x5809bcf5f3e87070	0x92584a68198c0286 0xd8b34f463af4b03c
0b001	0xfb0b271a781b4e27 0xf625c898230bb934	0x7d7b0db350f67ff6 0xf60db0dcb07eb1b1	0x3e2b1b189fbc10b4 0xe97d268ae2681267	0xe2ee9eaaa4ec5479 0x3cd6dc8228c5488d	0x92584a68198c0286 0xd8b34f463af4b03c	0xb455818159de0818 0x5809bcf5f3e87070
0b010	0xe2ee9eaaa4ec5479 0x3cd6dc8228c5488d	0x3e2b1b189fbc10b4 0xe97d268ae2681267	0x7d7b0db350f67ff6 0xf60db0dcb07eb1b1	0xfb0b271a781b4e27 0xf625c898230bb934	0x70e4228e70a3f8ff 0x9cc19db7de935d05	0x5eaaa2f0e48ef187 0x982cdffcf13dfb43
0b011	0x3e2b1b189fbc10b4 0xe97d268ae2681267	0xe2ee9eaaa4ec5479 0x3cd6dc8228c5488d	0xfb0b271a781b4e27 0xf625c898230bb934	0x7d7b0db350f67ff6 0xf60db0dcb07eb1b1	0x5eaaa2f0e48ef187 0x982cdffcf13dfb43	0x70e4228e70a3f8ff 0x9cc19db7de935d05
0b100	0xb455818159de0818 0x5809bcf5f3e87070	0x92584a68198c0286 0xd8b34f463af4b03c	0x70e4228e70a3f8ff 0x9cc19db7de935d05	0x5eaaa2f0e48ef187 0x982cdffcf13dfb43	0x7d7b0db350f67ff6 0xf60db0dcb07eb1b1	0xfb0b271a781b4e27 0xf625c898230bb934
0b101	0x92584a68198c0286 0xd8b34f463af4b03c	0xb455818159de0818 0x5809bcf5f3e87070	0x5eaaa2f0e48ef187 0x982cdffcf13dfb43	0x70e4228e70a3f8ff 0x9cc19db7de935d05	0xfb0b271a781b4e27 0xf625c898230bb934	0x7d7b0db350f67ff6 0xf60db0dcb07eb1b1
0b110	0x70e4228e70a3f8ff 0x9cc19db7de935d05	0x5eaaa2f0e48ef187 0x982cdffcf13dfb43	0xb455818159de0818 0x5809bcf5f3e87070	0x92584a68198c0286 0xd8b34f463af4b03c	0xe2ee9eaaa4ec5479 0x3cd6dc8228c5488d	0x3e2b1b189fbc10b4 0xe97d268ae2681267
0b111	0x5eaaa2f0e48ef187 0x982cdffcf13dfb43	0x70e4228e70a3f8ff 0x9cc19db7de935d05	0x92584a68198c0286 0xd8b34f463af4b03c	0xb455818159de0818 0x5809bcf5f3e87070	0x3e2b1b189fbc10b4 0xe97d268ae2681267	0xe2ee9eaaa4ec5479 0x3cd6dc8228c5488d

Apple's Customization on PAC Hardware

Key Access

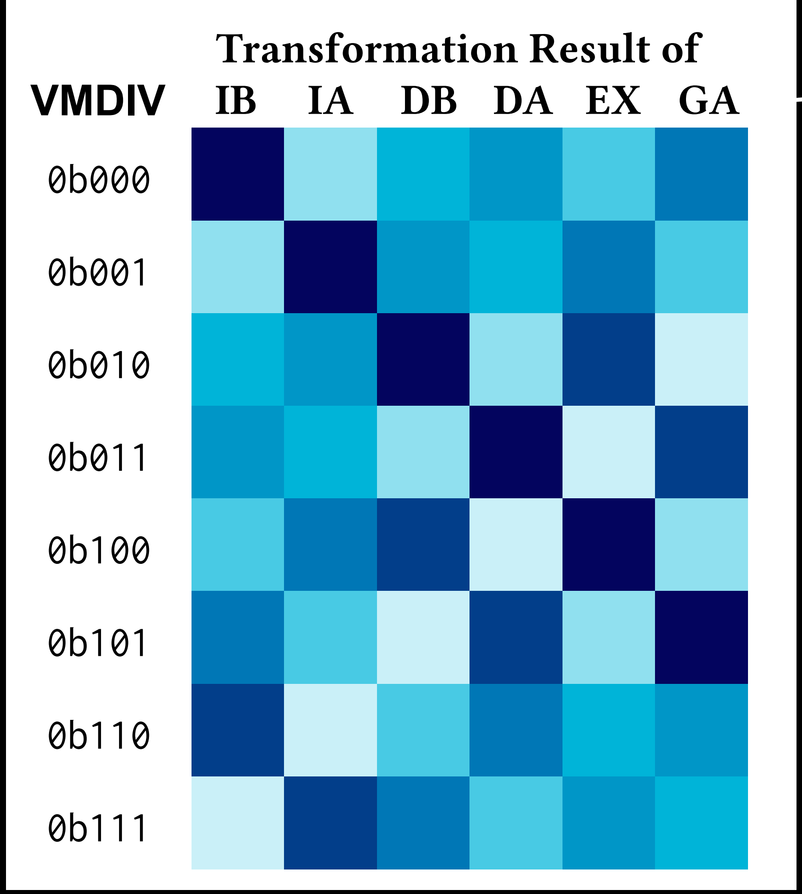
I set the VMDIV from 0b000 to 0b111



Apple’s Customization on PAC Hardware

Key Access

There are six **per-key salts** for differentiating Key Trans



Per-key-type Salt of					
IB	IA	DB	DA	EX	GA
0	1	2	3	4	5
1	0	2	3	4	5
2	3	0	1	6	7
3	2	1	0	7	6
4	5	6	7	0	1
5	4	7	6	1	0
6	7	4	5	2	3
7	6	5	4	3	2

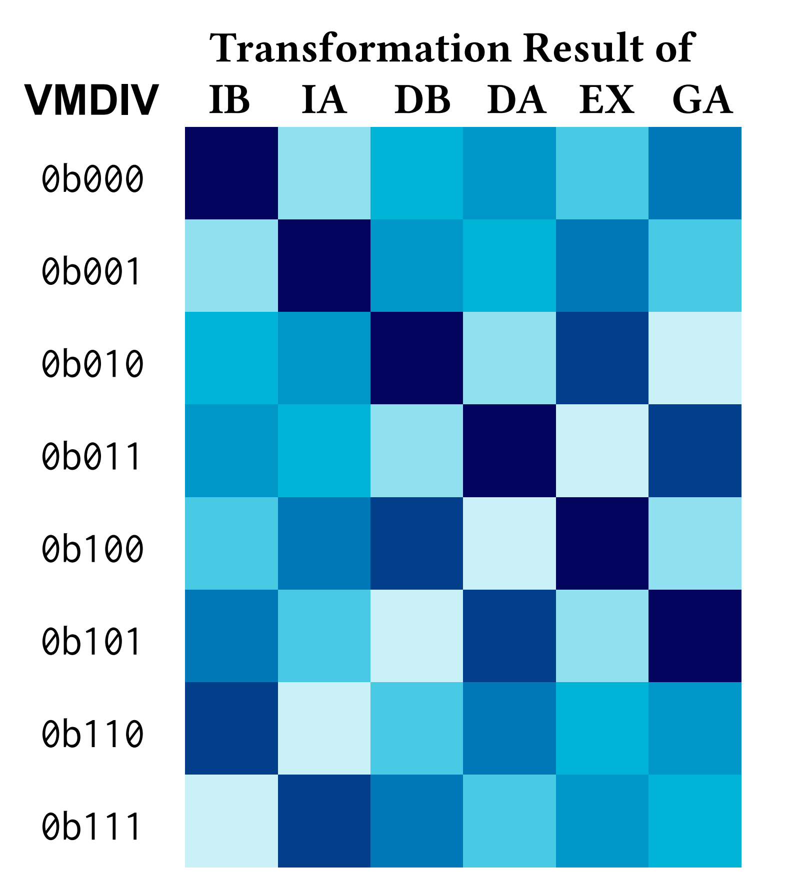
Only 8 combinations of per-key salt that XOR with VMDIV will produce the same symmetry

Apple's Customization on PAC Hardware

Key Access

per-key salt ⊕ VMDIVLO_EL2

is one of the inputs for Key Trans



Apple's Customization on PAC Hardware

Key Transformation

Inputs

- APKeyLo Register
- Operator of `msr APKeyHi_EL1, X1`
- per-key salt ⊕ VMDIVLO_EL2
- VMDIVHI_EL2

Output

- 128-bit PAC Key

Apple's Customization on PAC Hardware

Key Transformation

- Also deployed on EL2
- A **per-boot diversifier** for differentiating the Key Trans of different CPU Boots

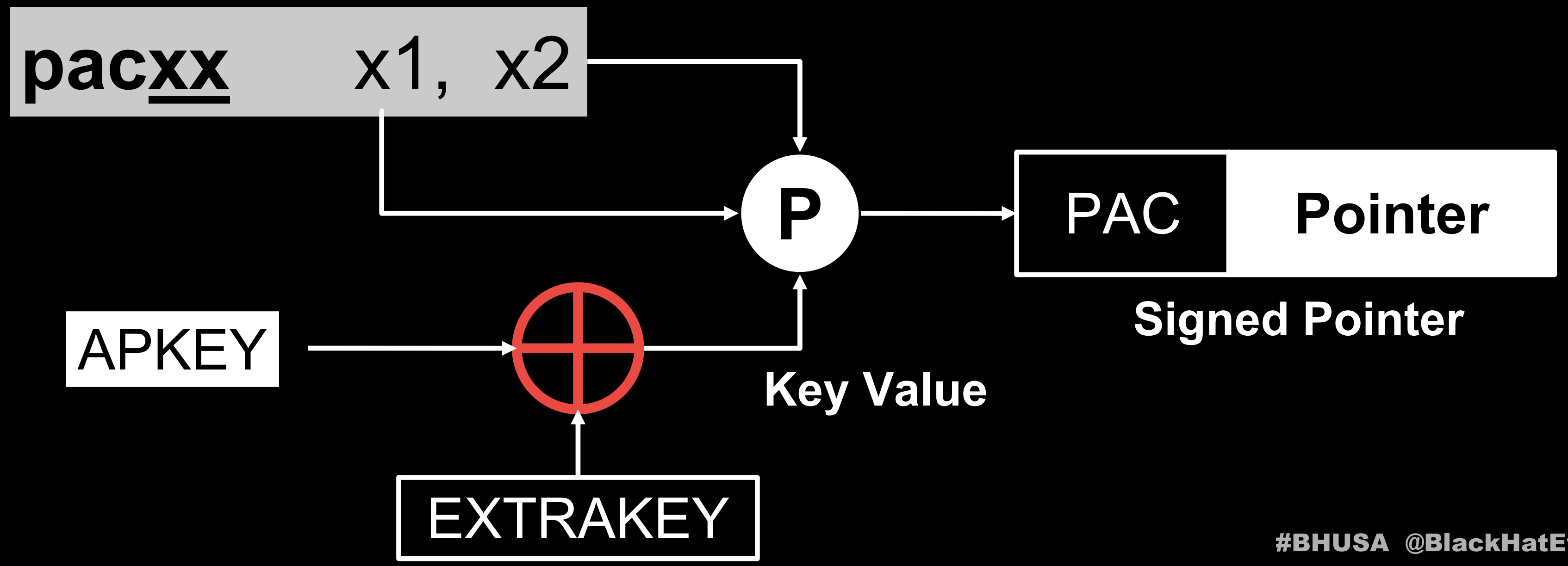
Apple's Customization on PAC Hardware

PAC/AUT

- A new 128-bit Key: **EXTRAKEY_EL1 (also Key Trans) XOR with APKEY** before PAC computation
- Enabled by **APCTL_EL1**

  bit[1]: Kernel

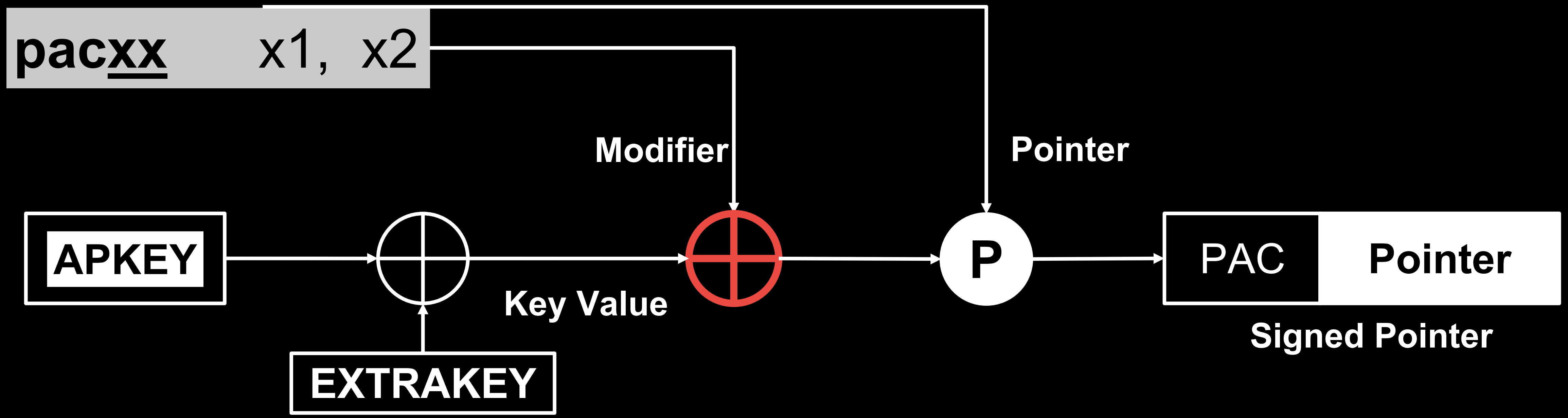
  bit[4]: User



Apple's Customization on PAC Hardware

PAC/AUT

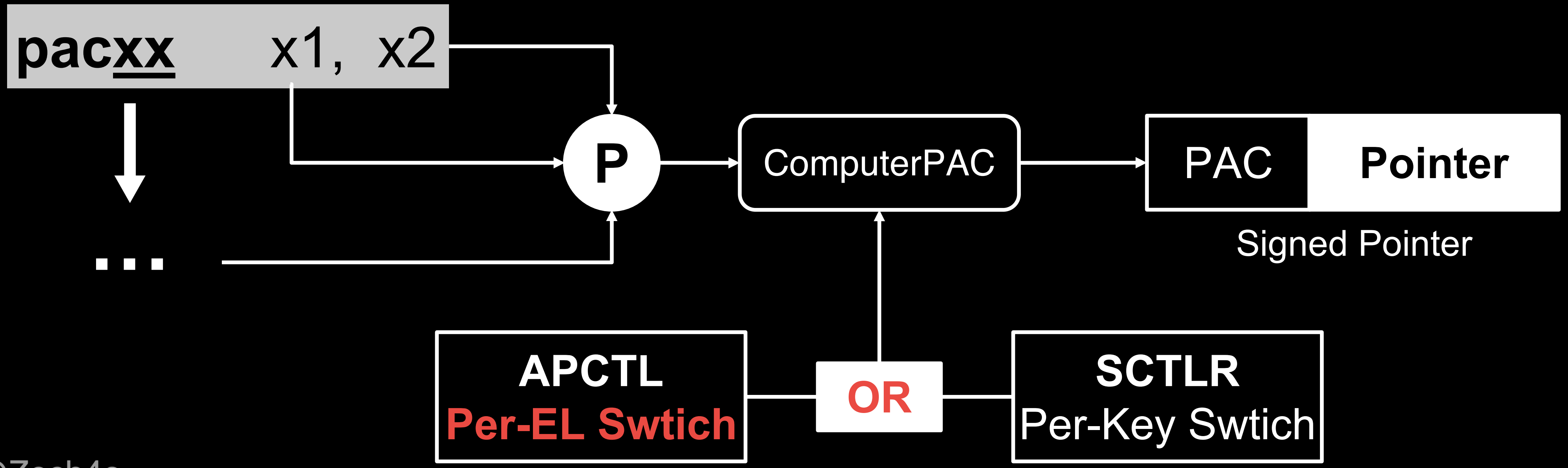
- PAC Algorithm is not QARMA
- (**Modifier ⊕ Key Value**) is one of the inputs



Apple's Customization on PAC Hardware

PAC/AUT

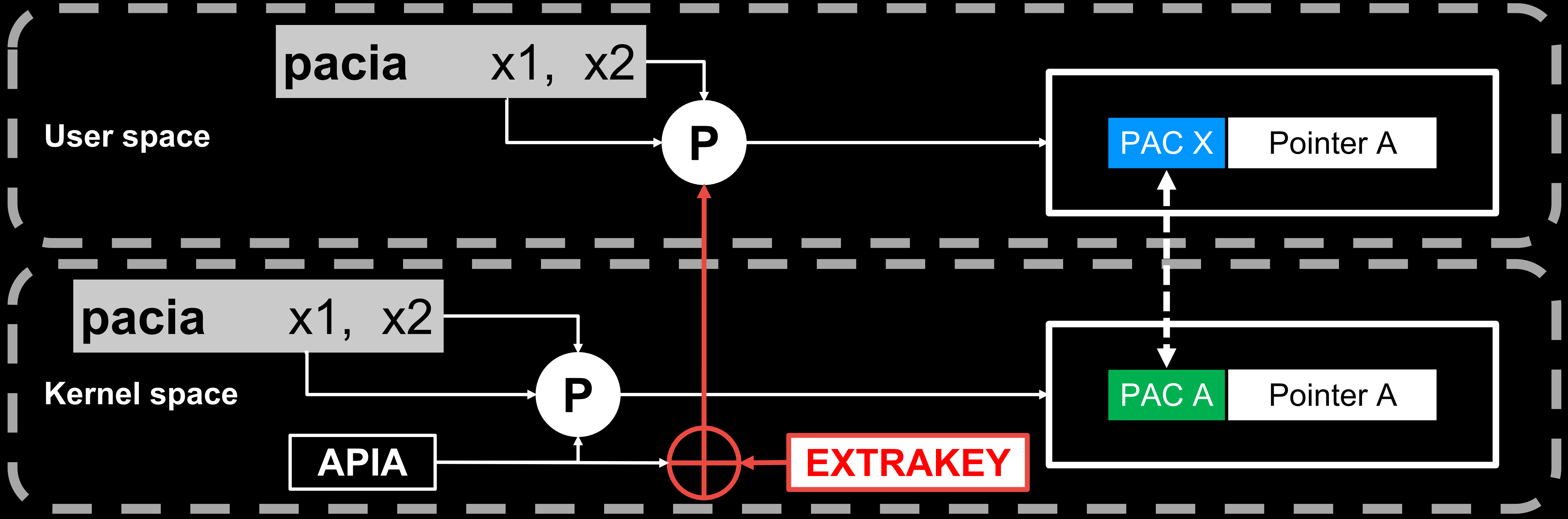
- A new **Per-EL switch** for PAC computation
- APCTL_EL1 bit[3]: Kernel; bit[2]: User



Cross-domain Attack Mitigation

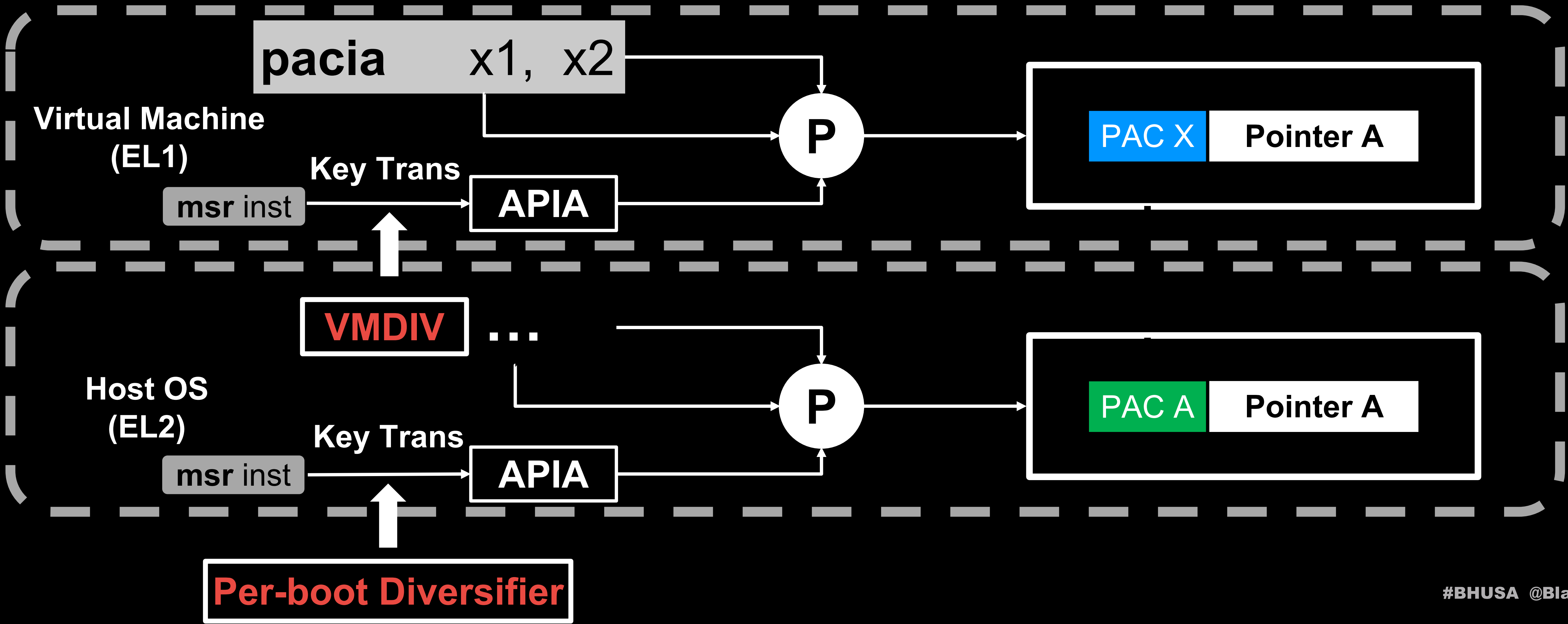
Cross-EL Attack Mitigation

XNU Kernel only enable EXTRAKEY on User space



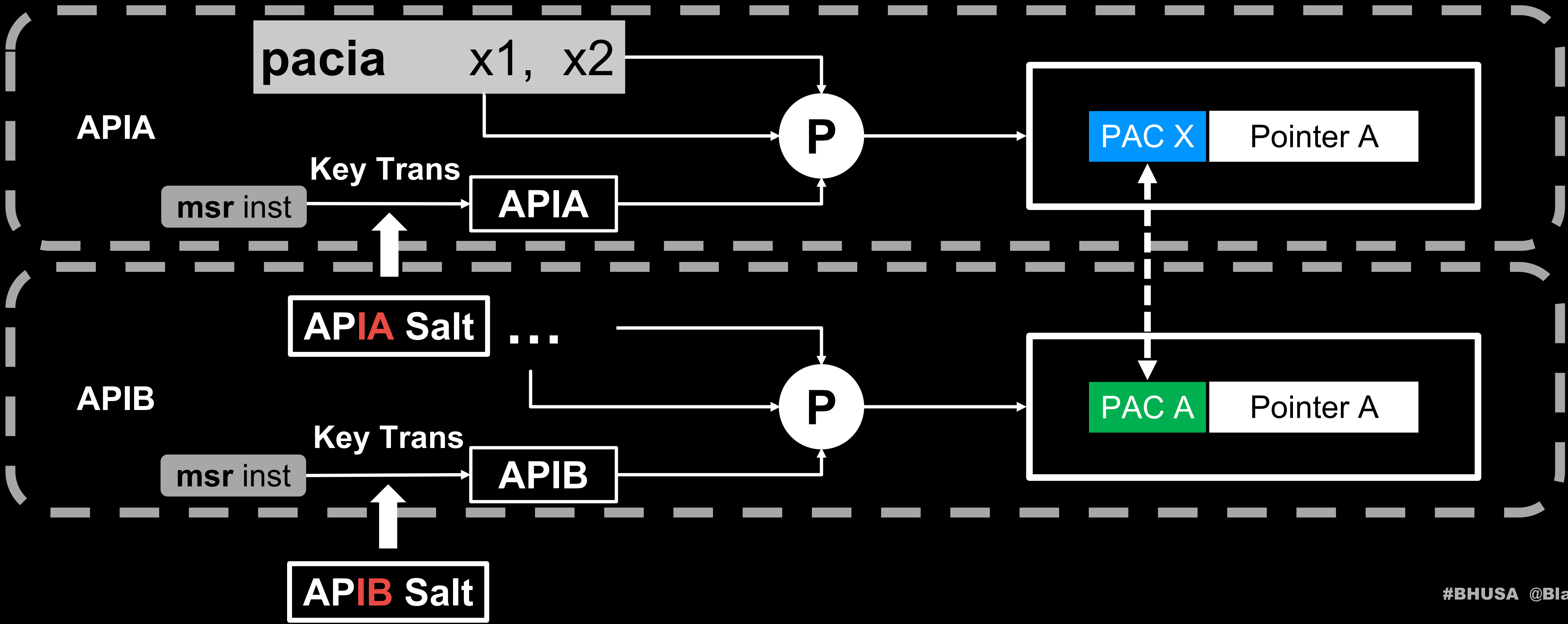
Cross-domain Attack Mitigation

Cross-VM/Boot Attack Mitigation



Cross-domain Attack Mitigation

Cross-Key Attack Mitigation



Key Management in the XNU Kernel

PAC Key Configuration

- Global (Static Value): APIA/DA/GA
- Per-Process: APIB/DB, EXTRAKEY

Key	APIA	APDA	APGA	APIB	APDB	EXTRAKEY
Scope	Global	Global	Global	Per-Process	Per-Process	Per-Process

Key Management in the XNU Kernel

PAC Instruction Scope

- pacia/da/ga: Global in Kernel, Per-Process in User

	pacia	pacda	pacga	pacib	pacdb
User (arm64e)	Per-Process	Per-Process	Per-Process	Per-Process	Per-Process
User (Non-arm64e)	-	-	Per-Process	Per-Process	-
Kernel	Global	Global	Global	Per-Process	Per-Process

Key Management in the XNU Kernel

PAC Instruction Scope

- pacia/da/ga: Global in Kernel, Per-Process in User
- pacib/db: Per-Process

	pacia	pacda	pacga	pacib	pacdb
User (arm64e)	Per-Process	Per-Process	Per-Process	Per-Process	Per-Process
User (Non-arm64e)	-	-	Per-Process	Per-Process	-
Kernel	Global	Global	Global	Per-Process	Per-Process

Key Management in the XNU Kernel

PAC Instruction Scope

- pacia/da/ga: Global in Kernel, Per-Process in User
- pacib/db: Per-Process
- Always Enable Kernel PAC (Per-EL Switch), Disable User PAC (IA/DA/DB) for non-arm64e process by disabling Per-Key switch

	pacia	pacda	pacga	pacib	pacdb
User (arm64e)	Per-Process	Per-Process	Per-Process	Per-Process	Per-Process
User (Non-arm64e)	-	-	Per-Process	Per-Process	-
Kernel	Global	Global	Global	Per-Process	Per-Process

Still Unknown

What's the algorithm used for Key Transformation?

Also, what's the PAC algorithm?

How Apple implements the per-boot diversifier?

- Maybe we can look into (RE) iBoot/SEP.

Summary

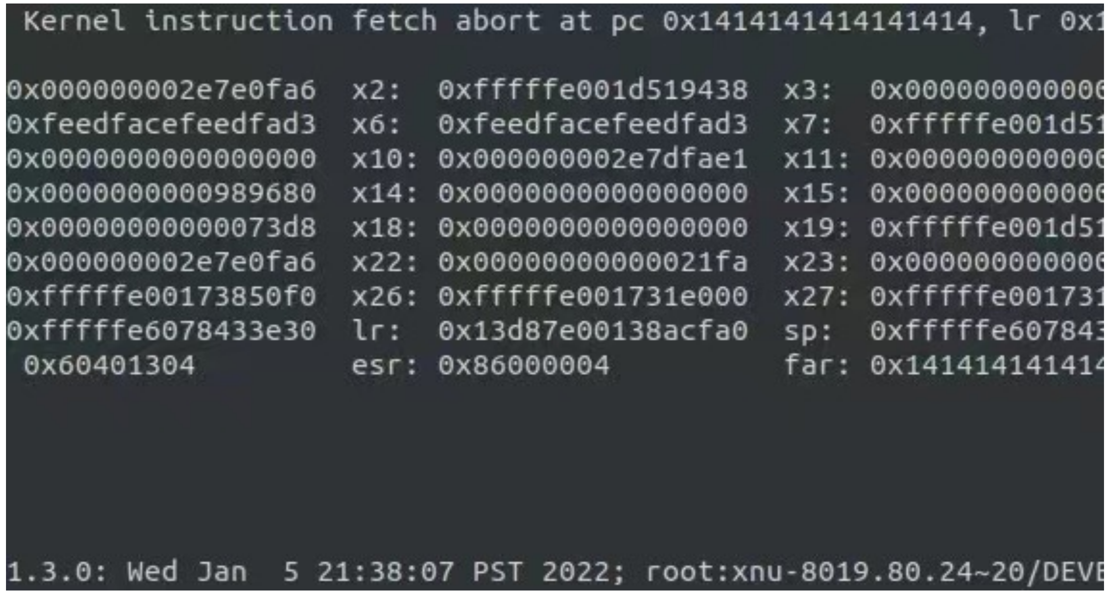
- Although there are some implementation remain unknown, the Design is clear.

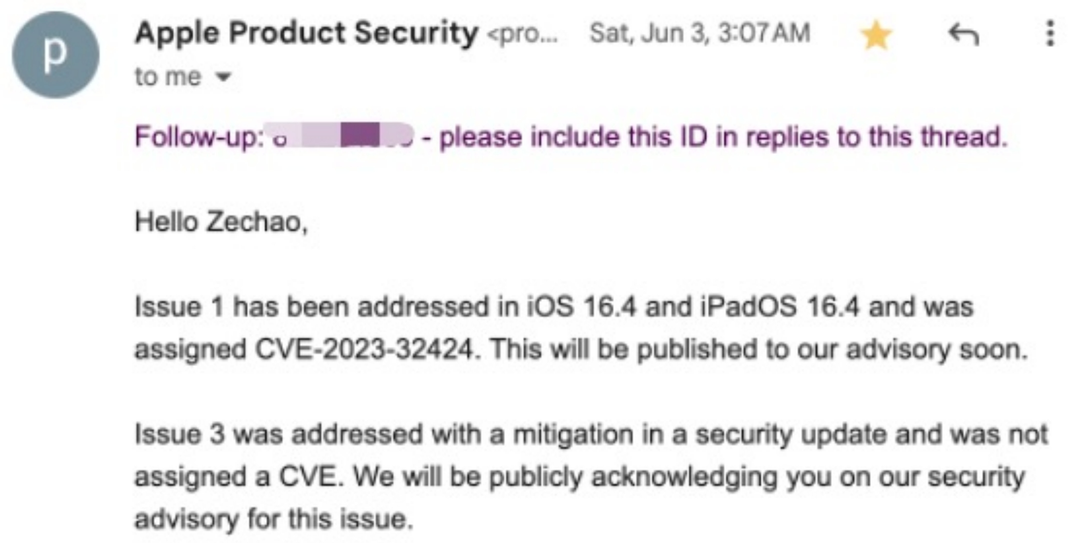
- Apple's PAC design looks simple, but insightful

- For ARM CPU Vendors and ARM, Apple give a solution to improve PAC

One More Thing

- I did a security analysis of kernel PAC protection.
- Got a CVE-2023-32424 for kernel PAC bypass from Apple.
- Check out my USENIX Security '23 paper
    - Demystifying Pointer Authentication on Apple M1
    - https://www.usenix.org/conference/usenixsecurity23/presentation/cai-zechao




Apple Product Security <pro... Sat, Jun 3, 3:07 AM
to me

Follow-up: [redacted] - please include this ID in replies to this thread.

Hello Zechao,

Issue 1 has been addressed in iOS 16.4 and iPadOS 16.4 and was assigned CVE-2023-32424. This will be published to our advisory soon.

Issue 3 was addressed with a mitigation in a security update and was not assigned a CVE. We will be publicly acknowledging you on our security advisory for this issue.

Thank you

Contacts

Zechao Cai - @Zech4o

zech4o@outlook.com