Demystifying Pointer Authentication on Apple M1

Zechao Cai[1,2], Jiaxun Zhu[1,2], Wenbo Shen[1,2,] , Yutian Yang[1,2], Rui Chang[1,2], Yu Wang[3], Jinku Li[4], and Kui Ren[1,2]

[1]*Zhejiang University, Hangzhou, China*
[2]*ZJU-Hangzhou Global Scientific and Technological Innovation Center, Hangzhou, China*
[3]*Hangzhou Cyberserval Co., Ltd., Hangzhou, China*
[4]*Xidian University, Xi'an, China*

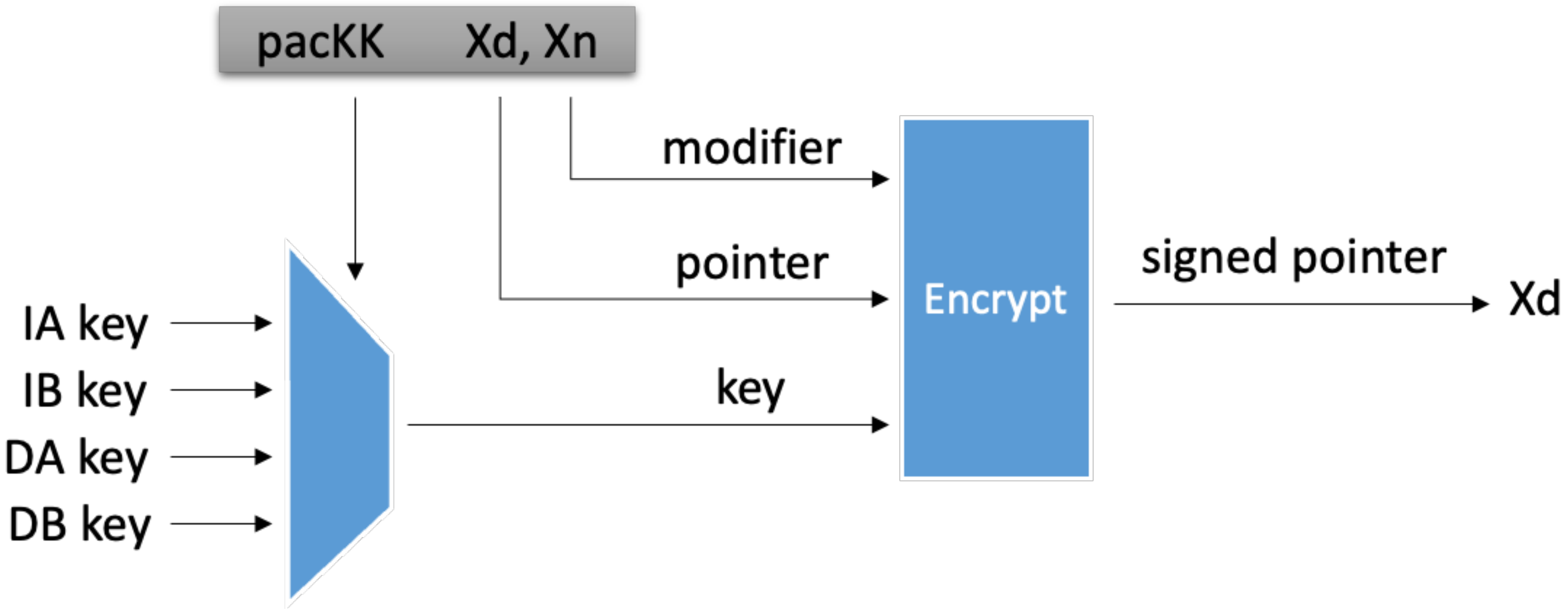
Outline

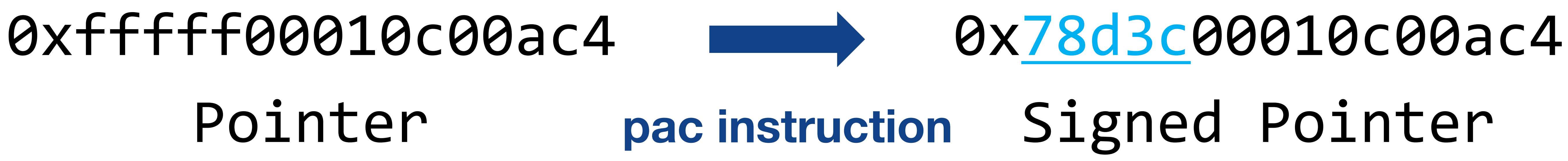
- Pointer Authentication
  - What is Pointer Authentication (PAC)
  - Current Research State of Apple PAC
- Our Motivation
  - Lack of systematic analysis of Apple PAC Protection
- Our Contribution
  - m1n1-based reverse engineering framework
  - Disclosure of Apple's hardware implementation
  - Comprehensive analysis of PA-based XNU kernel protection
- Our Findings
  - Apple's PA Hardware
  - PA-based XNU Kernel Protection
  - Security analysis

Pointer Authentication

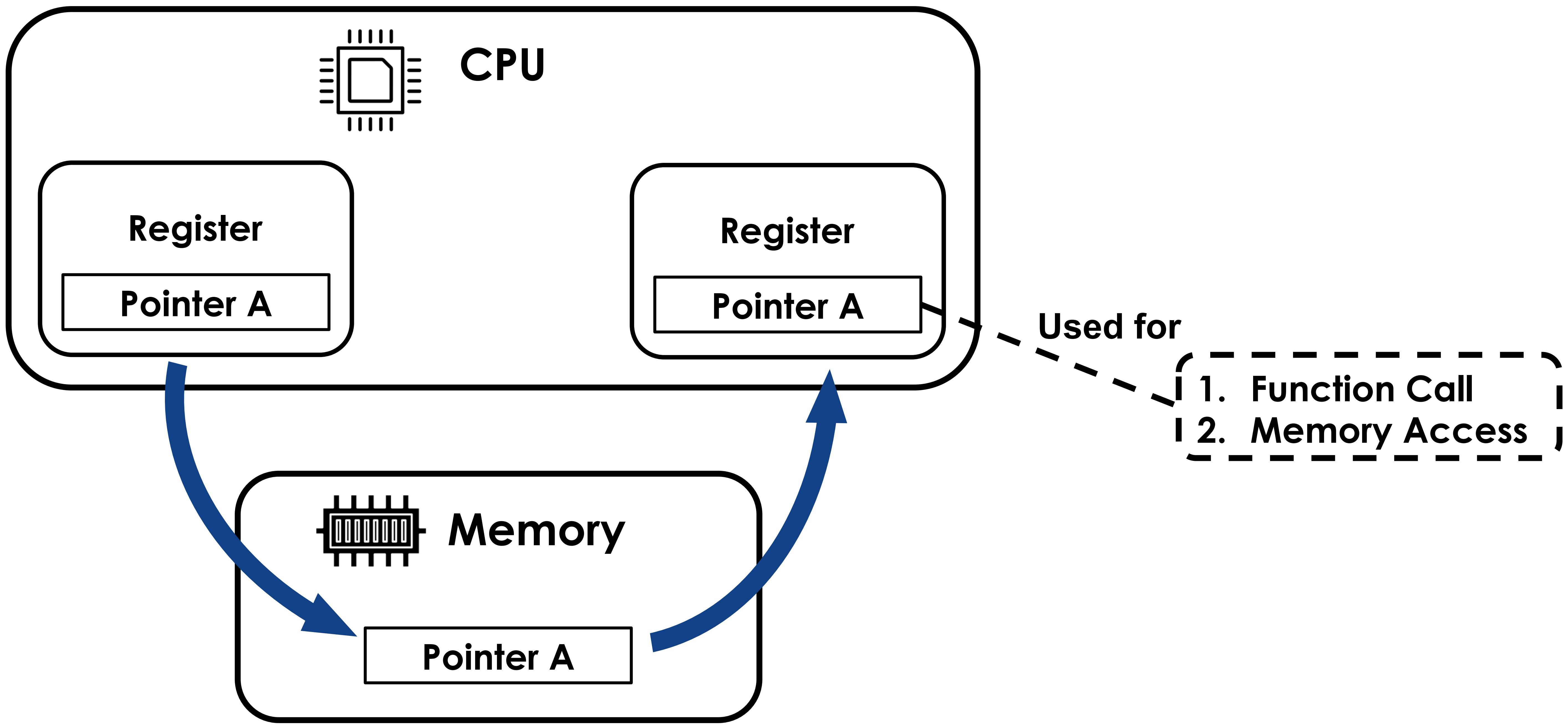
Pointer Authentication

- **ARMv8.3 Specification**
- **1 control register**
  - SCTLR_EL1.EnIA/IB/DA/DB)
- **5 key registers**
  - APIA/APIB/APDA/APDB/APGA
- **2 kinds of instructions**
  - pac*/aut*

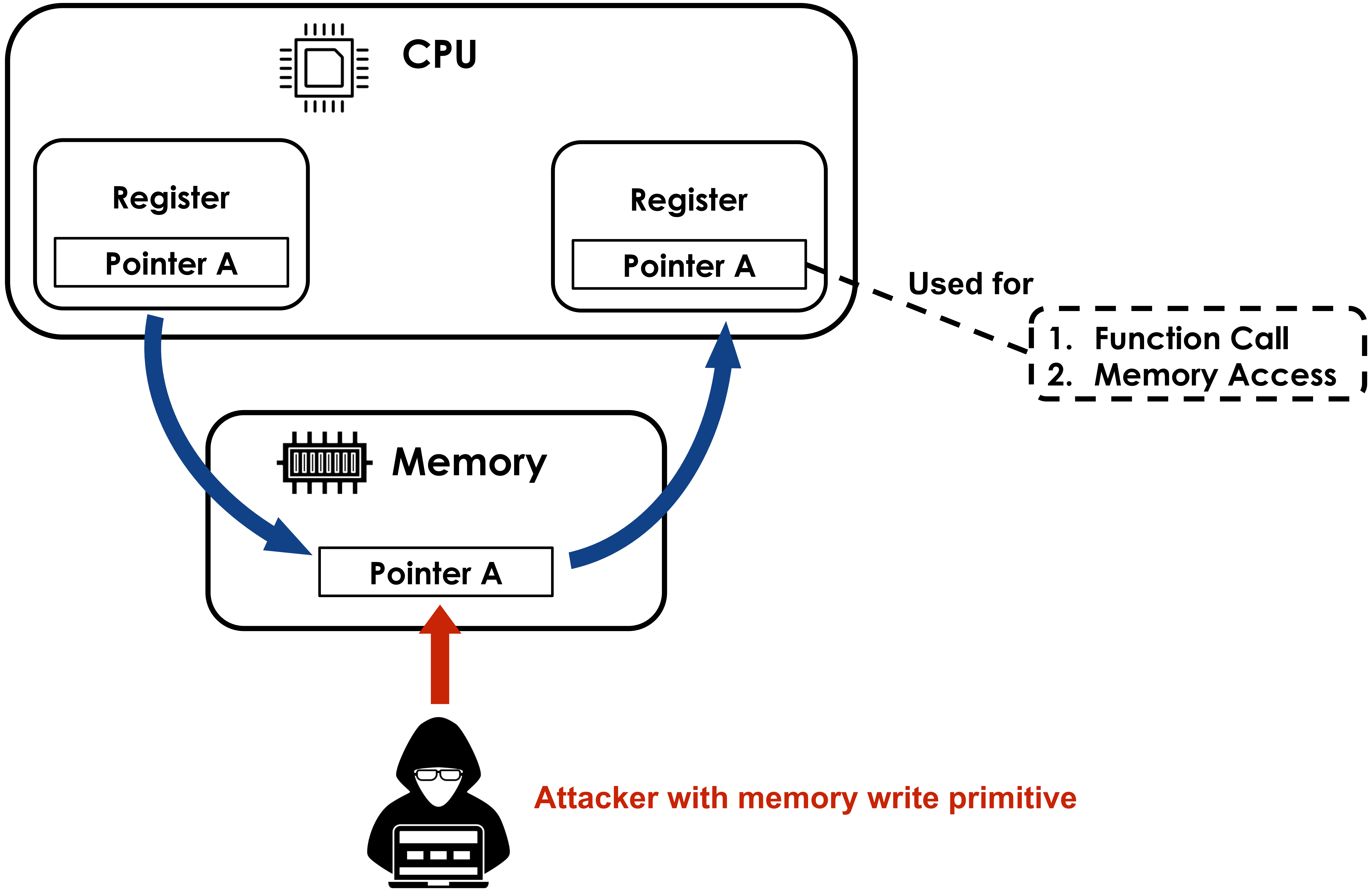




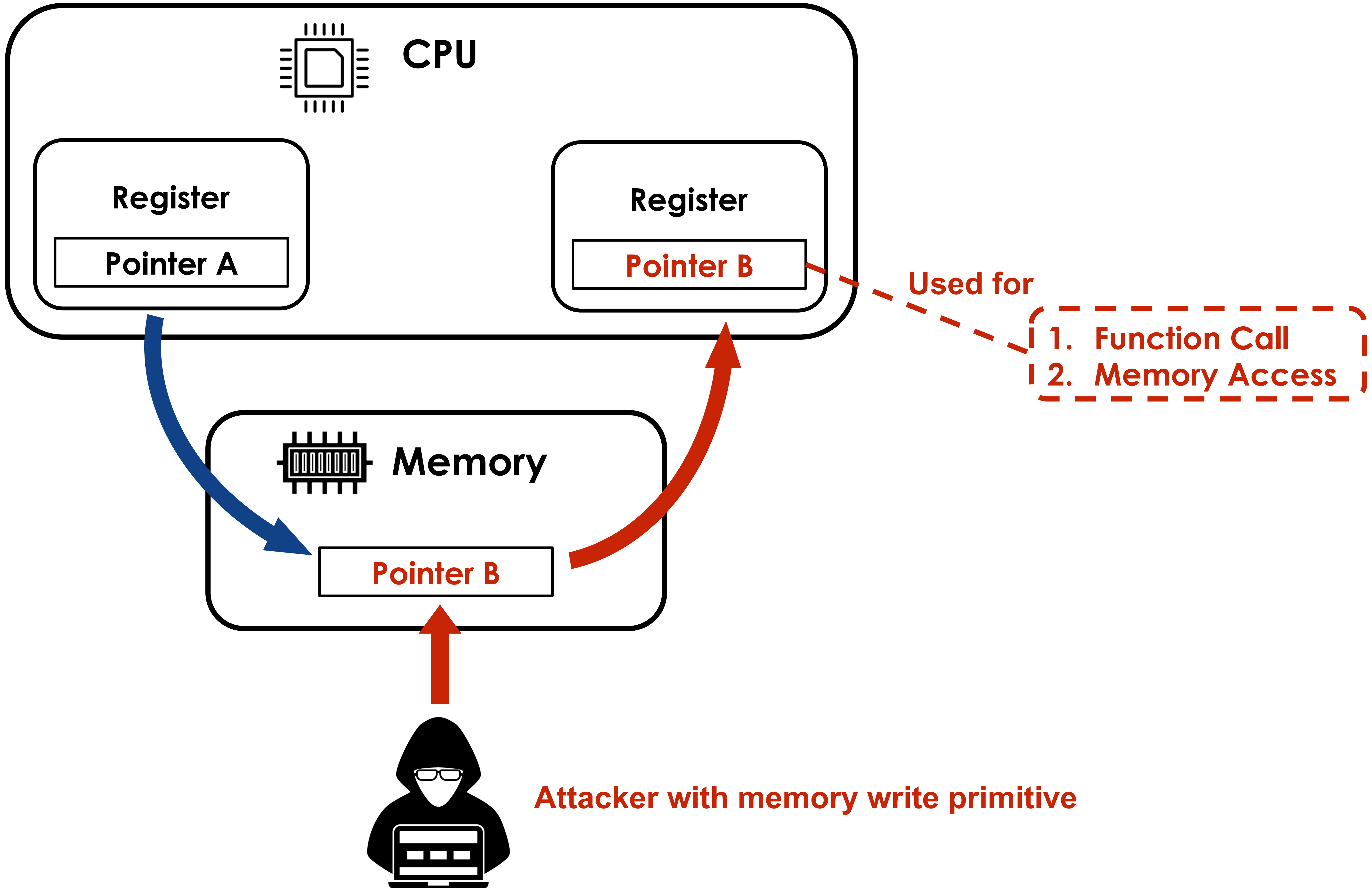
Pointer Authentication



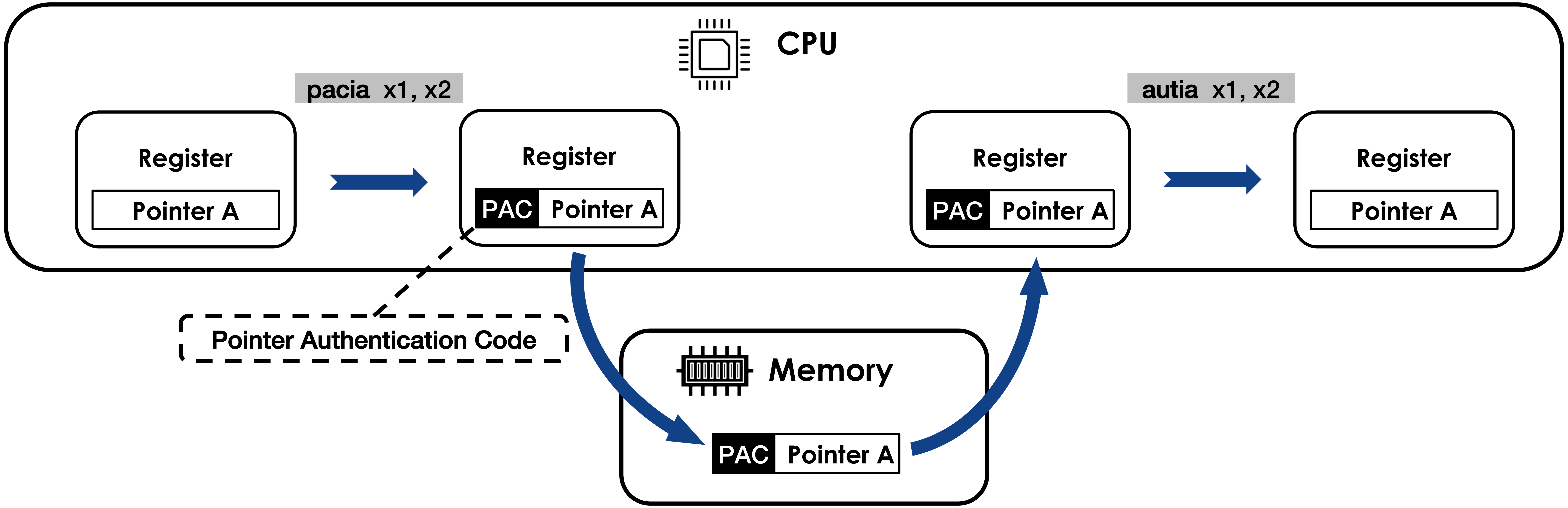
Pointer Authentication



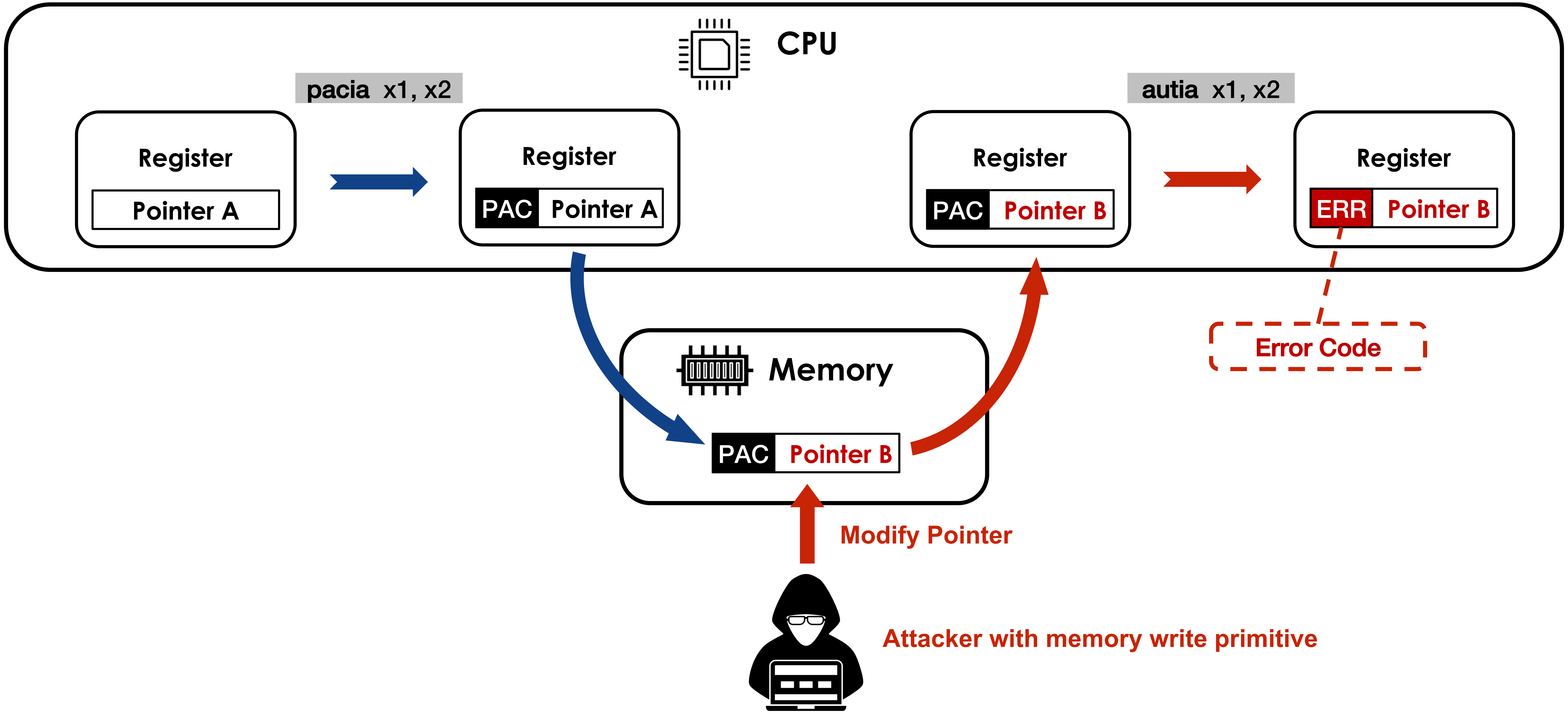
Pointer Authentication



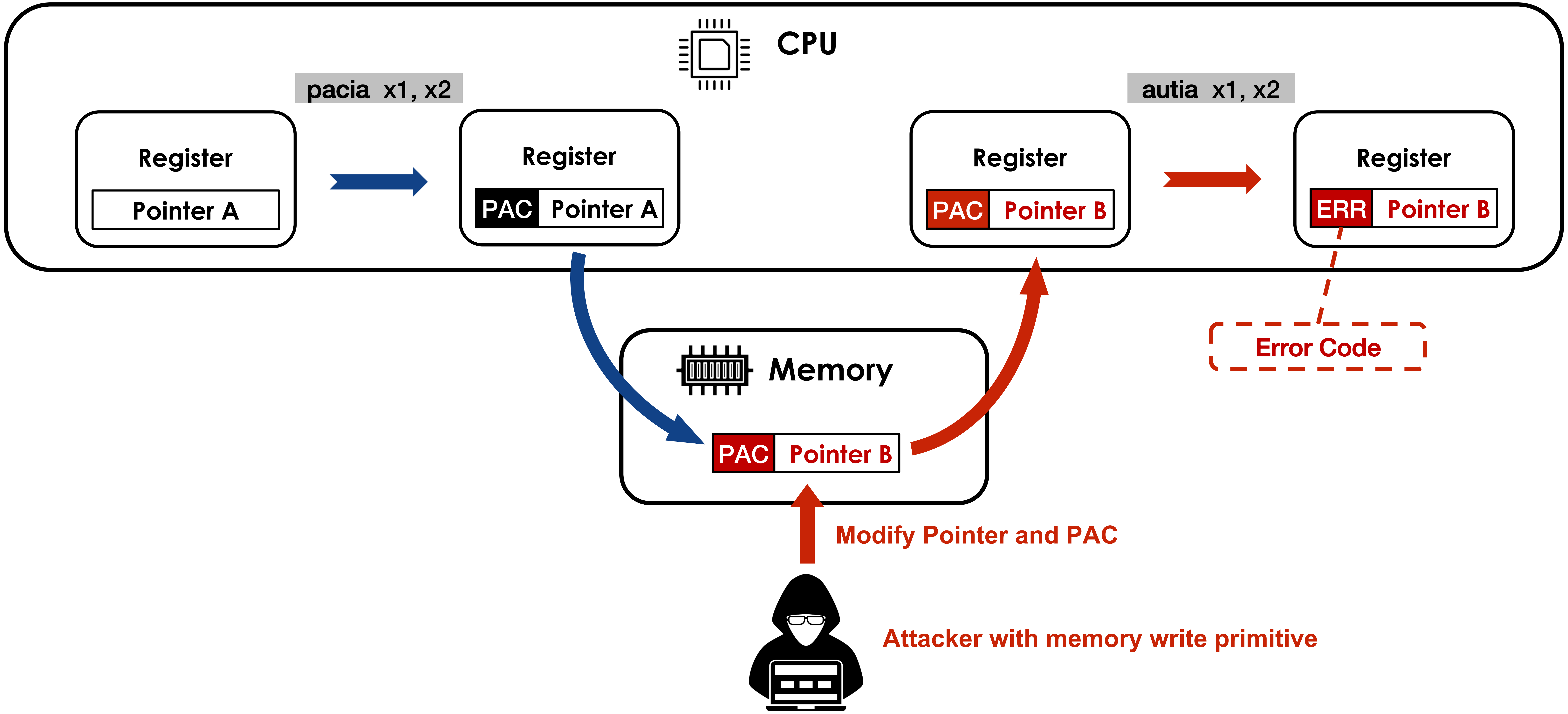
Pointer Authentication



Pointer Authentication



Pointer Authentication



Apple's PA Protection

- **Apple is the first one to implement and deploy PA hardware**
  - State-of-the-art mitigation against pointer corruption attack
  - Since its debut, the number of public kernel exploits has decreased

Apple PAC

Since A12 (iPhone XS, 2018)

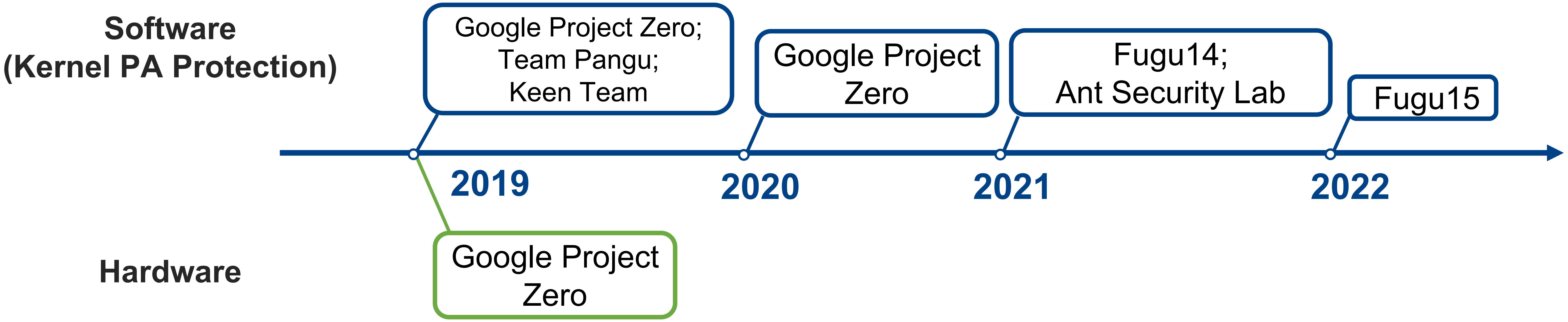
Feature	A10	A11, S3	A12, S4	A13, S5	A14, A15, S6, S7	M1 Family
Kernel Integrity Protection	✓	✓	✓	✓	✓	✓
Fast Permission Restrictions		✓	✓	✓	✓	✓
System Coprocessor Integrity Protection			✓	✓	✓	✓
Pointer Authentication Codes			✓	✓	✓	✓
Page Protection Layer		✓	✓	✓	✓	See Note below.

Apple SoC Security[1]

[1] Apple SoC Security. https://support.apple.com/guide/security/apple-soc-security-sec87716a080/web

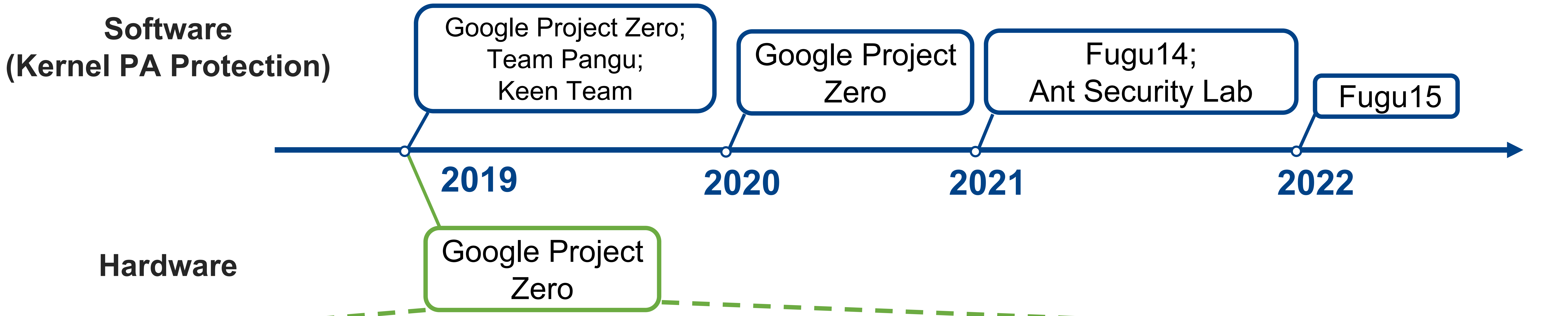
Our Motivation

- **Although there are a lot of research works on of Apple's PA**
  - Most of them focus on PA-based **software** protection
  - How does Apple implements Apple PA **hardware remain unknown**



Our Motivation

- **Although there are a lot of research works on of Apple's PA**
  - Most of them focus on PA-based **software** protection
  - How does Apple implements Apple PA **hardware remain unknown**



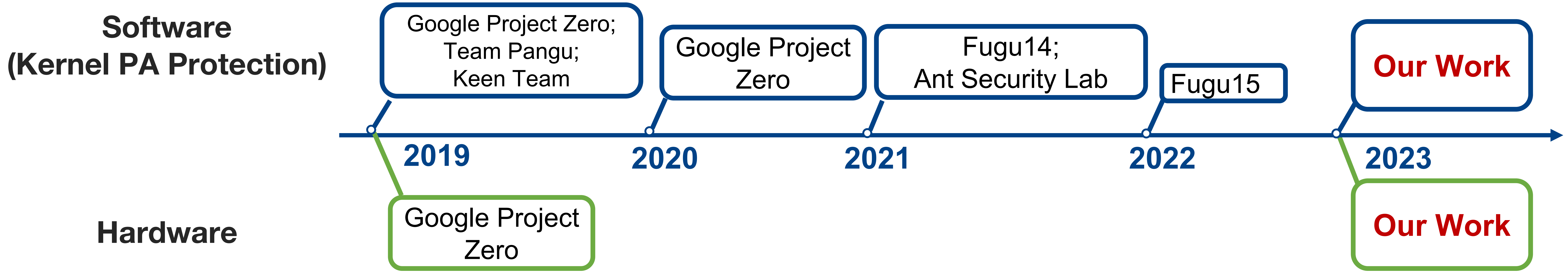
Brandon Azad :

> It seemed that the A12 manages to break either cross-EL PAC symmetry or cross-key PAC symmetry.

> pointer read from kernel memory. These two values differed despite the fact that the PAC keys should be the same in userspace and the kernel, so once again it appeared that the A12 was conjuring some sort of dark magic.

Our Motivation

- The **hardware** implementation remains **unknown**
- As a result, the PA **software can not be analyzed systematically**
- There is still **no systematic analysis** of Apple PA hardware and software



Our Contribution

- We build a **reverse engineering framework** based on m1n1 (an open-sourced hypervisor) to analyze PA hardware and software on Apple M1
- We reveal **how Apple customizes the PA hardware** to introduce undisclosed security properties — Cross-domain (Cross-EL/Key/VM/Boot) attack mitigation
- We analyze the implementation of PA-based XNU kernel protection and identify four attack surfaces. **Apple acknowledged our findings publicly, fixed these issues in a security update and assigned us a CVE.**

m1n1-based RE Framework

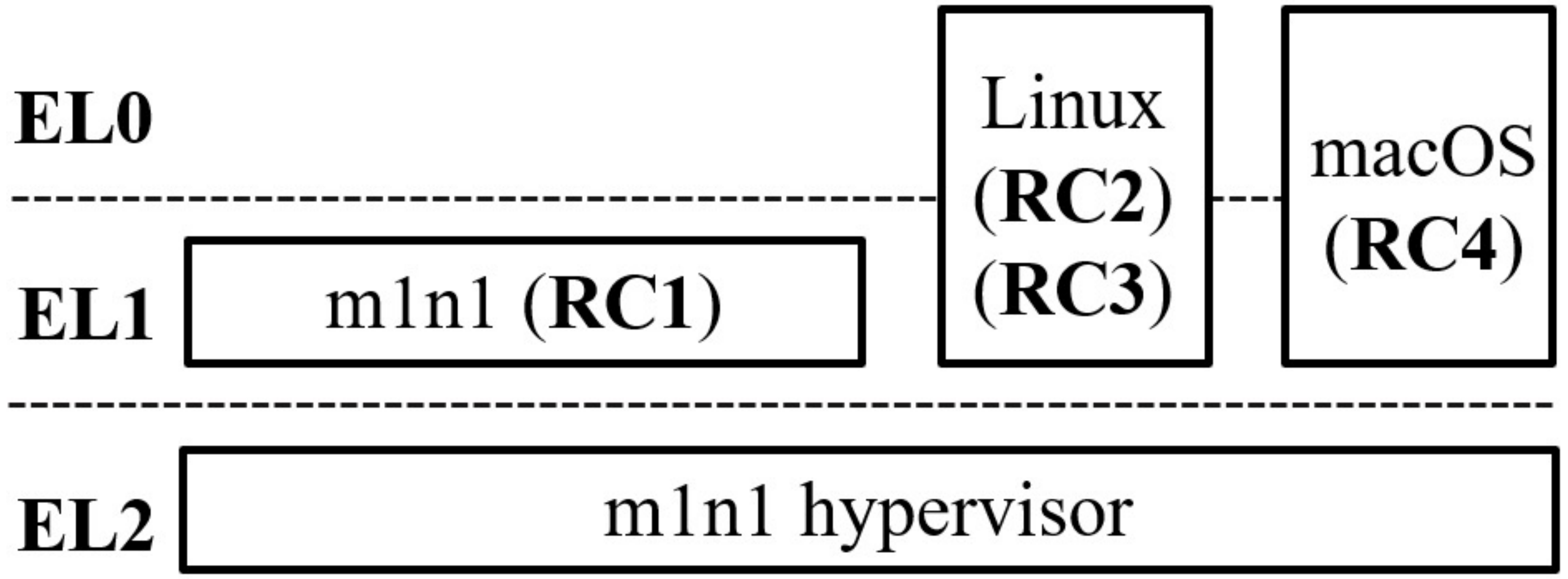
- **Required Capabilities (RC)**

RC1: Identify undisclosed PA-related Apple-specific system registers

RC2: Read/Write actual PA key values

RC3: Profile the undisclosed PA instruction behavior

RC4: Debug the XNU kernel dynamically



m1n1-based RE Framework

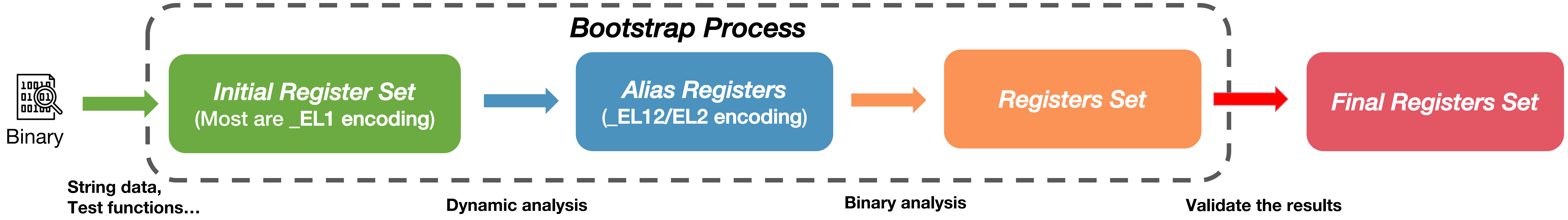
➢ **RC1: Identify** undisclosed PA-related Apple-specific system registers

Challenge: Apple introduced a lot of **undisclosed system registers**

Our solution: We identify registers based on

- **Binary** information

- **System Register Redirection Hardware Feature**



m1n1-based RE Framework

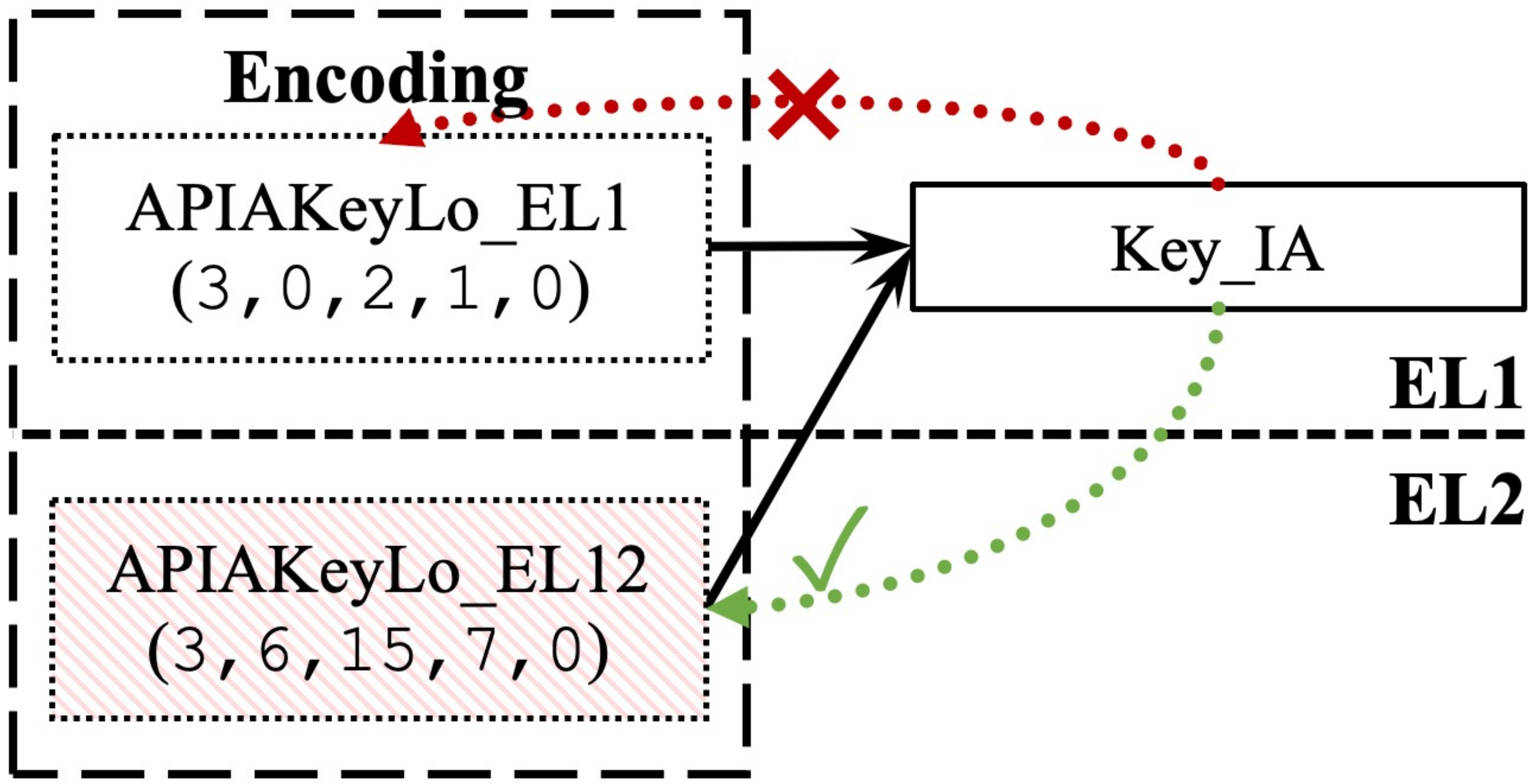
➢ **RC2: Read/Write** the actual key value

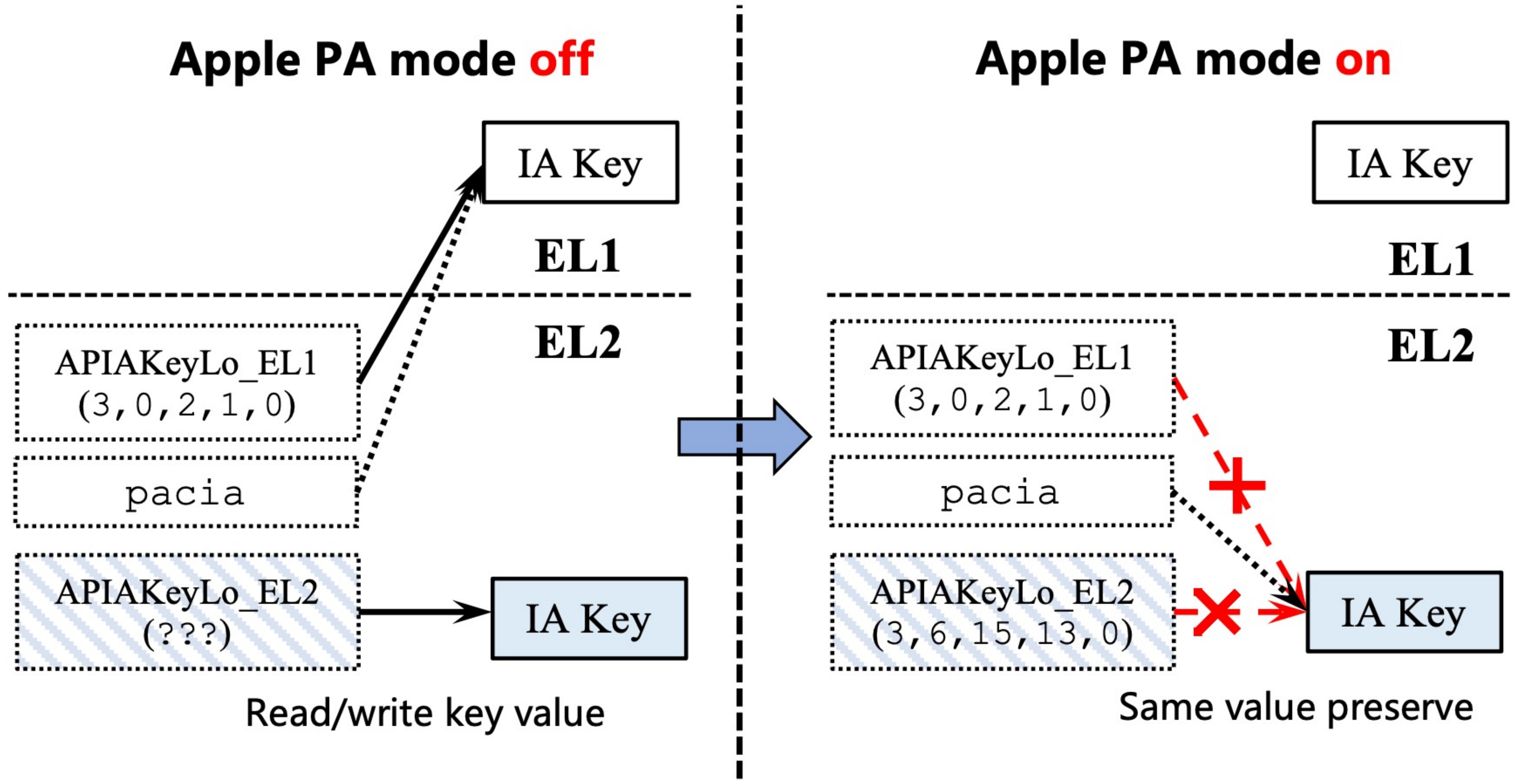
Challenge: Apple implemented a **hardware-based PAC key protection**

Our techniques:

- **For EL1 Key,** Read/Write the key **from EL2** exception level

- **For EL2 Key,** Read/Write the key **before Apple PA is enabled**



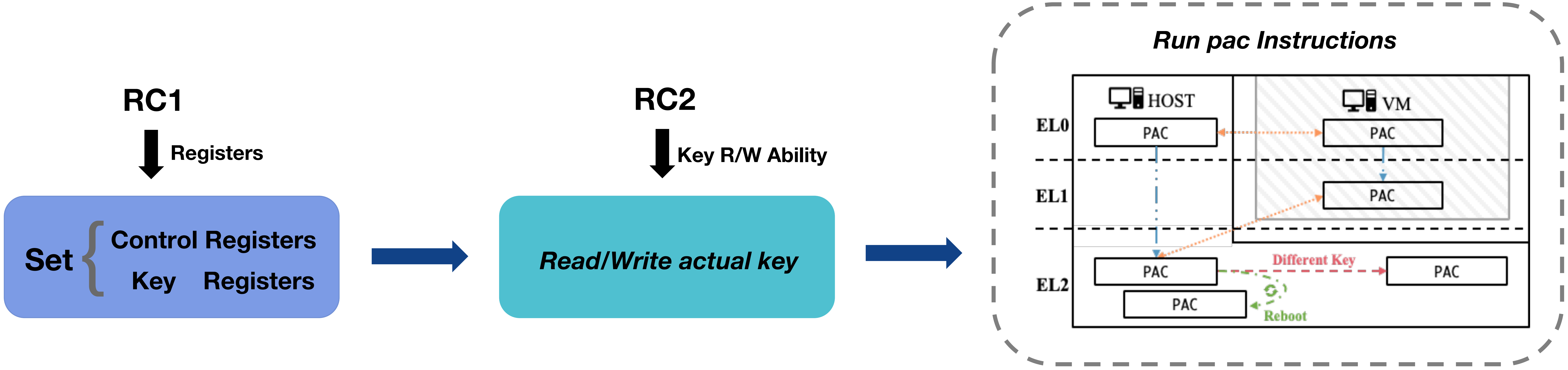


m1n1-based RE Framework

- **RC3: Profile** the undisclosed PA instruction behavior

Challenge: We need to analyze the **complex interplays** between registers and intructions

Our solution:



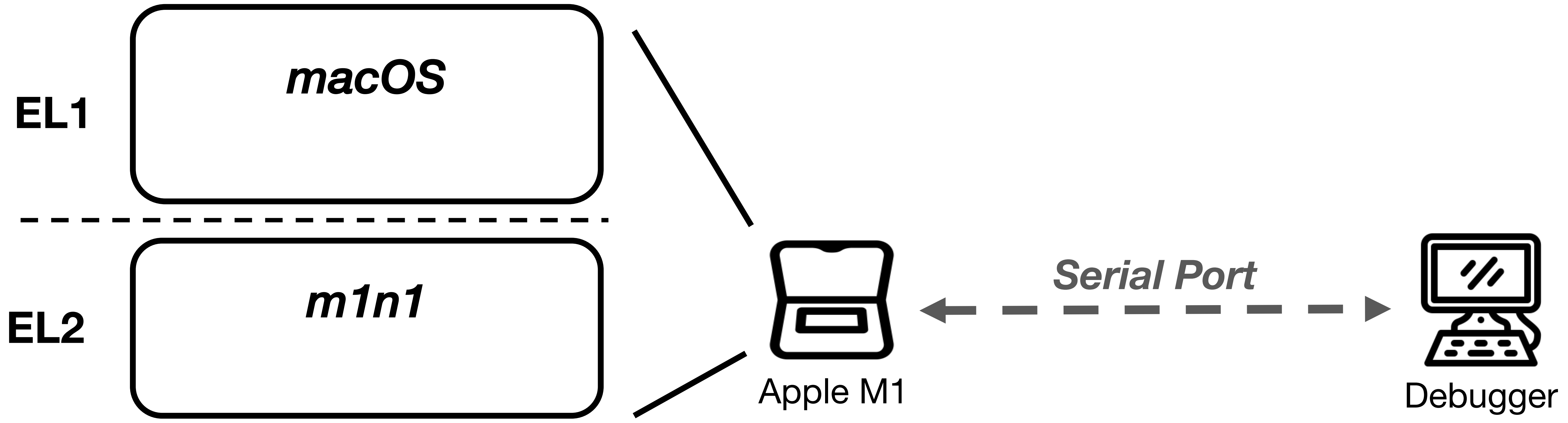
Step 1 | Step 2 | Step 3

m1n1-based RE Framework

➢ **RC4: Debug** the XNU kernel dynamically

Challenge: LLDB (provided by Apple) **does not support active kernel debugging** on Apple M1

Our solution: We implement active kernel debugging based m1n1 hypervisor



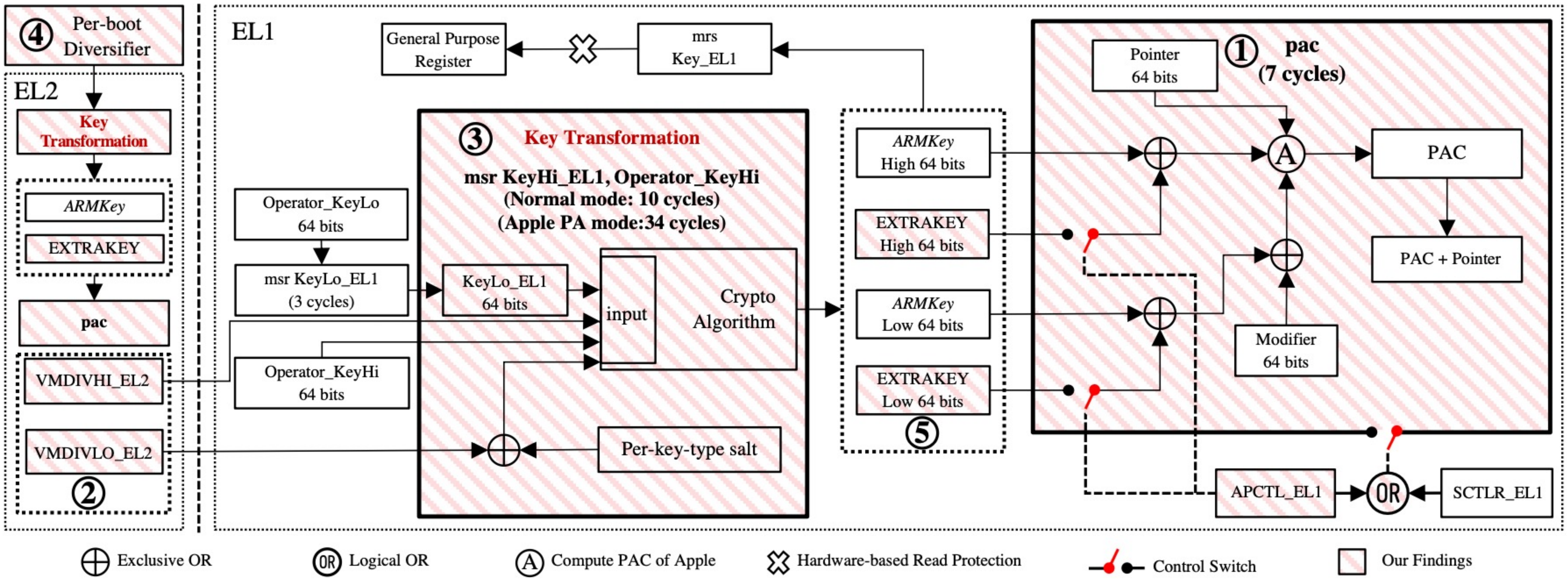
Our Findings

Apple's PA Hardware

- **Finding Overview**

  ① **Controllability, PAC algorithm**

  ②-⑤ **Cross-domain Attack Mitigation**



Apple's PA Hardware

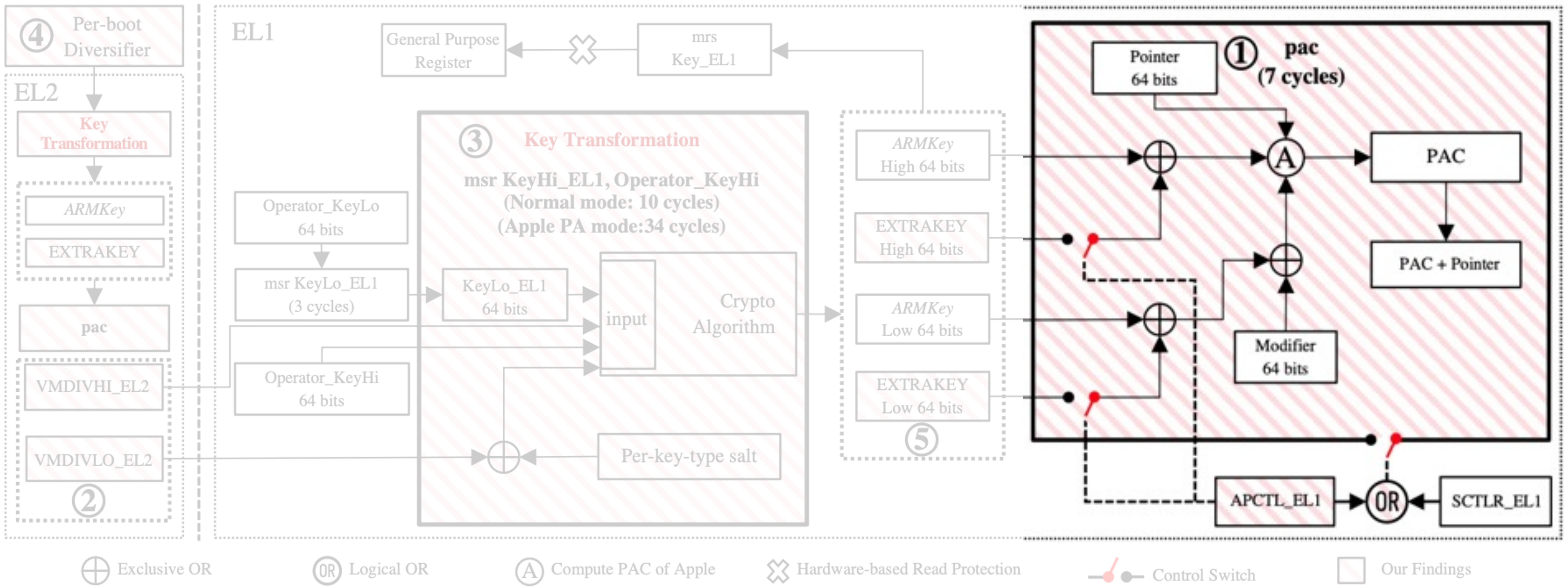
- **Controllability & PAC Algorithm ①**

`APCTL_EL1`

- `bit[0]`: Enable Apple PA
- `bit[2], bit[3]`: Enable PA on user (`bit[2]`) or kernel space (`bit[3]`)
- `bit[1], bit[4]`: Enable EXTRAKEY on user (`bit[4]`) or kernel space (`bit[1]`)

PAC Algorithm is not QARMA

- `(Modifier XOR KeyValue)` is one of the inputs



Apple's PA Hardware

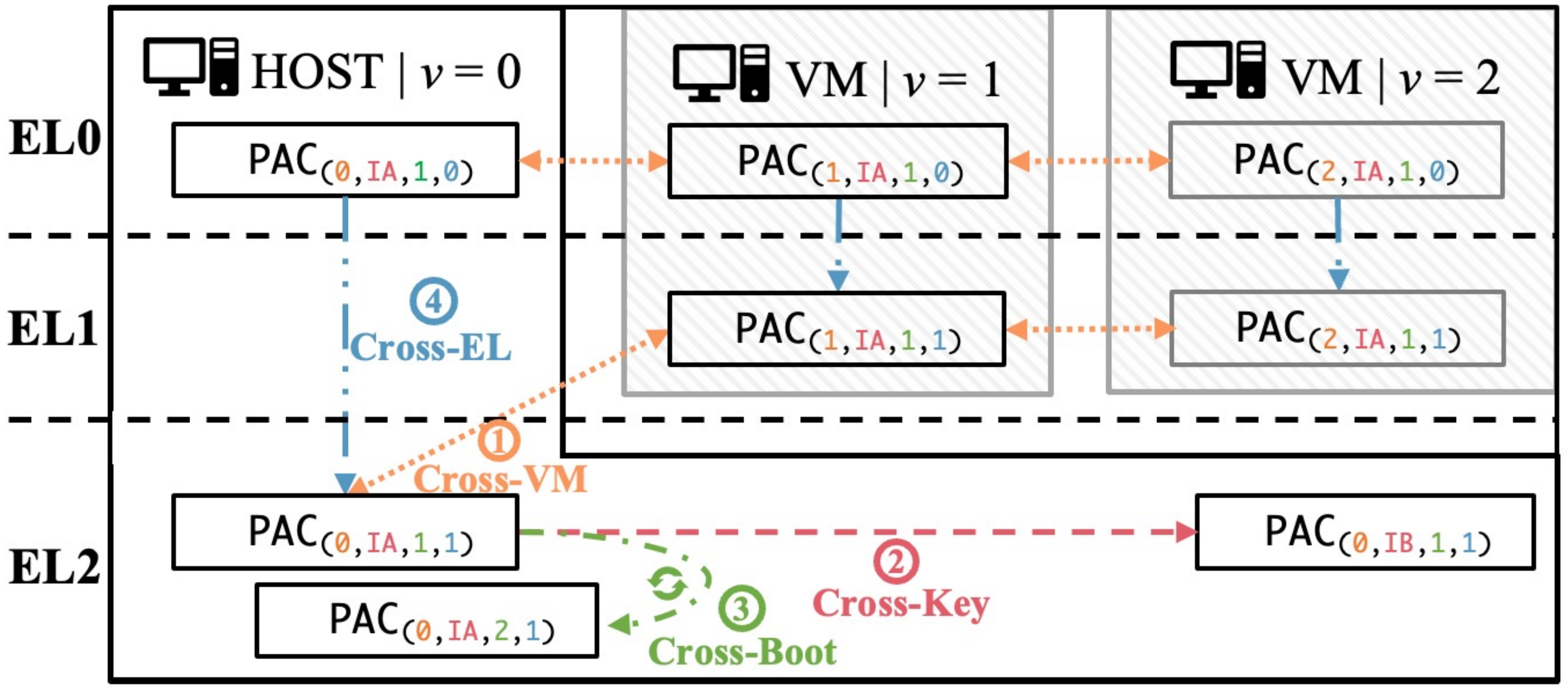
- **Cross-domain Attack**
  - ***Pointer substitution attack across different domains***

Cross-VM: From VM to Host and Other VMs

Cross-Key: E.g., From APIA-signed to APIB-signed

Cross-Boot: From Boot Round 1 to Round 2

Cross-EL: From User space to Kernel space



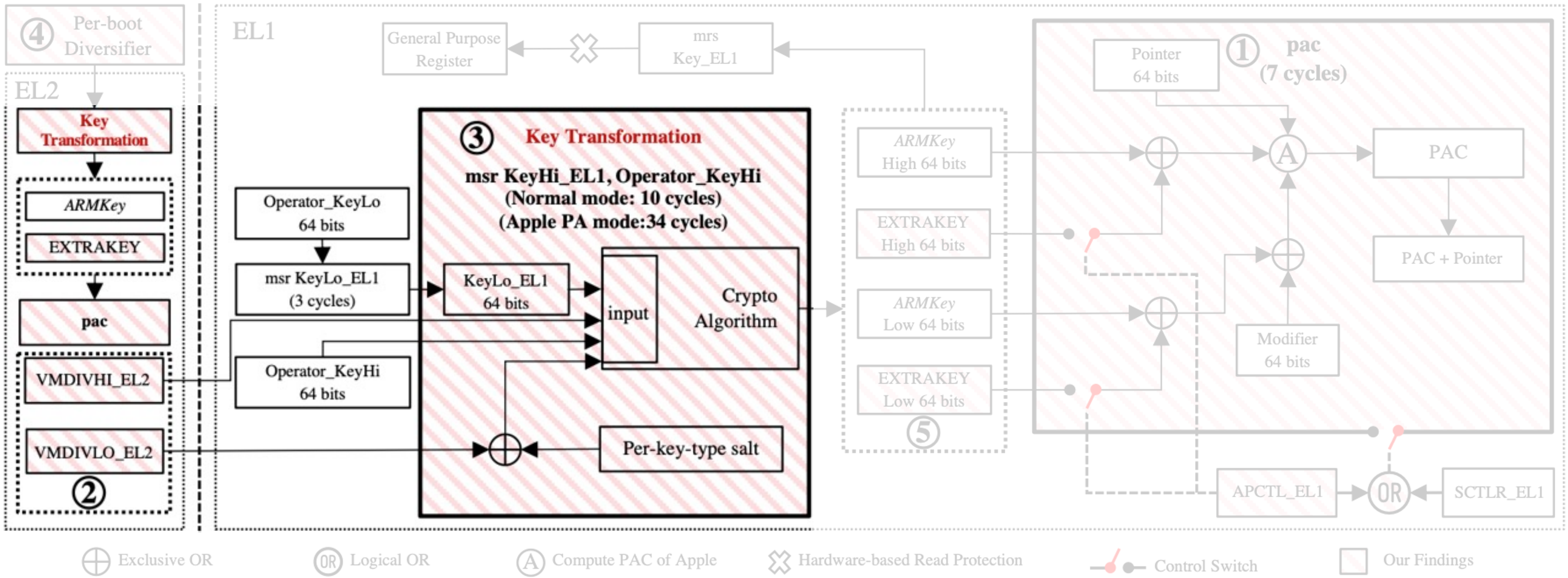
Formalization of Cross-domain Attack in the paper 😁

Apple's PA Hardware

- **Cross-VM attack mitigation**

Setting the higher 64-bit PAC Key will trigger a Key Transformation

② VMDIV_EL2 is used for differentiate the Key Transformation between VM and Host

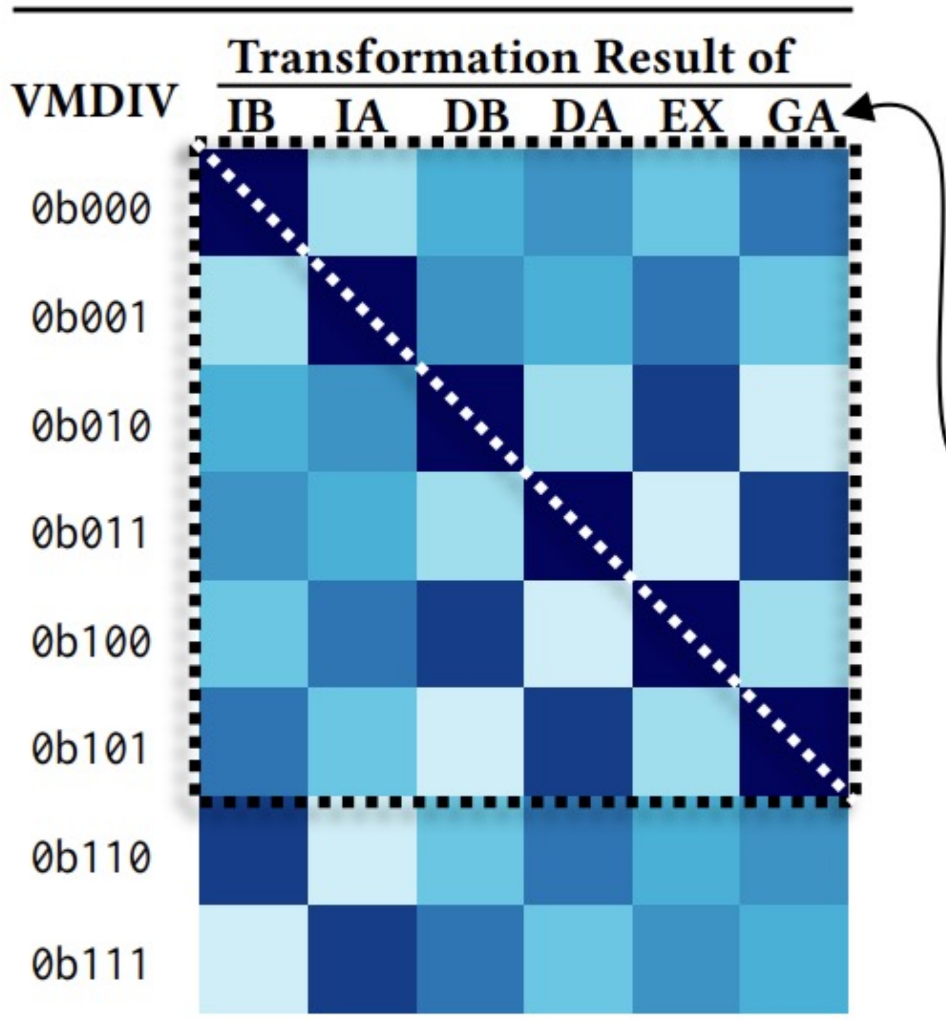


Apple’s PA Hardware

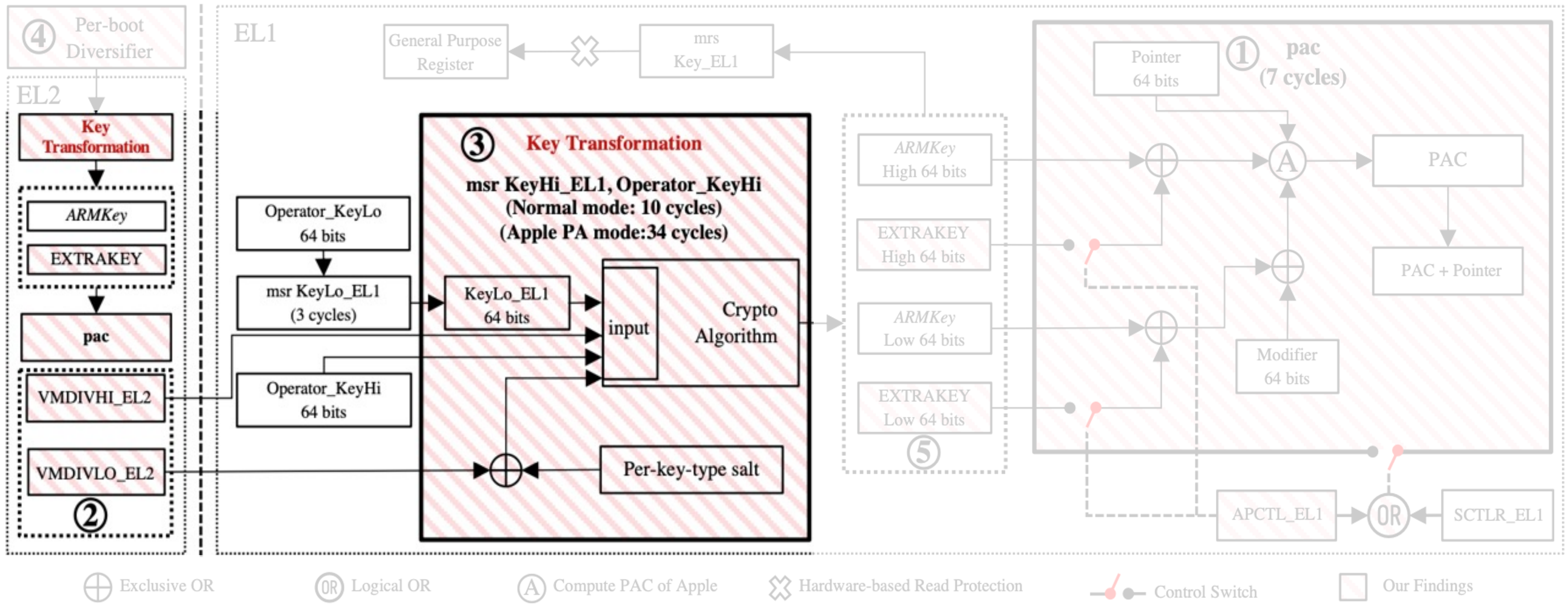
- **Cross-Key attack mitigation**

  Key Transformation introduces per-key-type salts to differentiate the results for different key types

  ③ VMDIV_EL2 ⊕ per-key-type salts is one of the inputs for Key Transformation



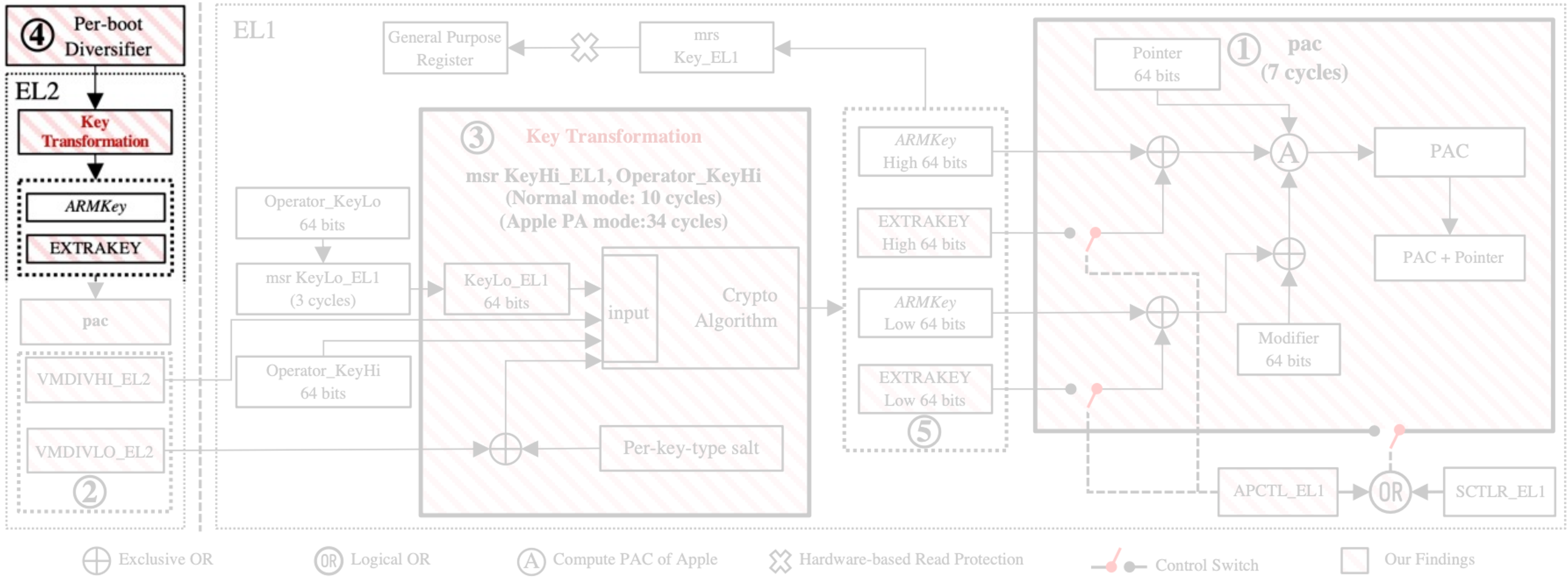
Per-key-type Salt of					
IB	IA	DB	DA	EX	GA
0	1	2	3	4	5
1	0	2	3	4	5
2	3	0	1	6	7
3	2	1	0	7	6
4	5	6	7	0	1
5	4	7	6	1	0
6	7	4	5	2	3
7	6	5	4	3	2



Apple's PA Hardware

- **Cross-Boot attack mitigation**

EL2 Key Transformation introduces ④ per-boot diversifier to differentiate the results for different CPU Boots



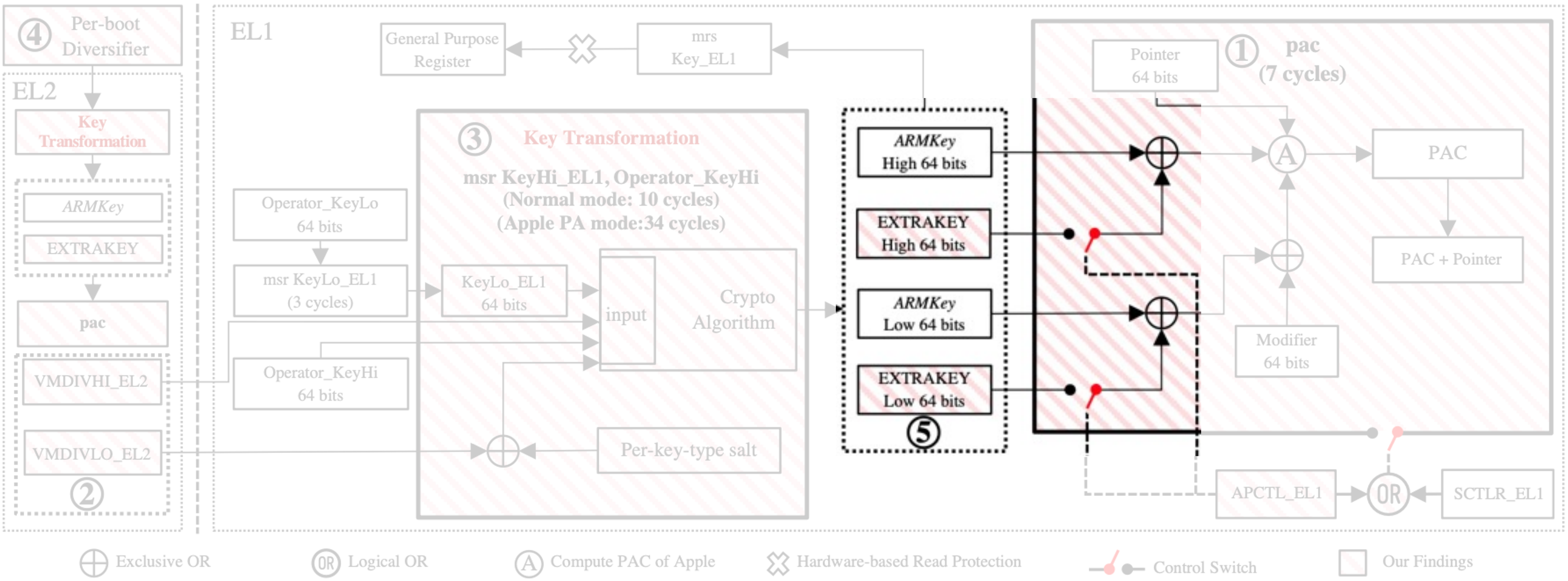
Apple's PA Hardware

- **Cross-EL attack mitigation**

Apple introduces an EXTRAKEY to differentiate the PAC computation between user and kernel space

⑤ EXTRAKEY ⊕ APKeys (APIA/IB/DA/DB/GA) is the actual key value for PAC computation

Controlled by APCTL_EL1 (bit[1]: Kernel, bit[4]: User (XNU Kernel only enable bit[4]))



PA-based Kernel Protection

- **Sign/Auth Interfaces Analysis**
  - We analyze **all pac instructions** in XNU kernel
  - The result shows that XNU kernel uses
    - 9 types of signing modifiers
      - (5 types in official documentation)
    - 6 policies for generating modifier constant
      - (2 policies in official documentation)

Key Usage	Modifier	Target
IA (762376)	Hash(function_type) (38564)	Function pointer
	Hash(function_name) + storage address(384)	Recovery Handler/ Corecrypto related
	Storage address (27568)	ppl_handler(89)
		Copy/destroy_helper_block (626)
		Block_invoke function(115)
		Ptrs in seg: __auth_ptr (25452)
		Block function pointer (1286)
	Hash(root_class, function_type, function_name) + Storage address (694596)	Vtable entry(694596)
	Zero (1264)	__chkstk_darwin* func (156)/ BluetoothFamily function (9)/ kext_weak_symbol_referenced/ Parameter func ptr (1098)
DA (31225)	Hash(data_field_type) + Storage addr (2645)	Proc0 (1)
		__NSConcreteGlobalBlock(115)
		_Block_descriptor (115)
		sysctl_oid_list (2414)
	Hash(data_pointer_name)+ Storage addr (464)	Data pointer
	Hash(root_class) + Storage addr (28116)	V-table pointer
IB(110852)	SP	Return address
DB	-	-
GA	Storage Address	Thread state/ Exception state/ Data Blob

PA-based Kernel Protection

- **Key Management**
  - **XNU kernel configures the keys**
    - Global: `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

PA-based Kernel Protection

- **Key Management**
  - **PAC instruction scope**
    - `pacia/da/ga:` global in kernel space, per-process in user space
    - `pacib/db:` per-process
    - For `non-arm64e` process, the XNU kernel disable the user space PAC

PAC instructions	`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

Security Analysis

- **We validate 4 attack surfaces (88 cases) and report them to Apple**

① **Incomplete Sensitive Data Identification**

- Potential Enhancements

② **Incomplete Interrupt Context Protection**

- **Fixed in a security update and acknowledged publicly**

③ **Signing Gadget**

- **Fixed and assigned a CVE (CVE-2023-32424)**

④ **Key Leakage**

- Potential Enhancements

AS \ Result	Identified	Validated	Fixed Cases
❶	153	83	6
❷	17+18*	2	2+18*
❸	1	1	1
❹	2	2	-

* 18 cases are identified in XNU-7195, and all of them are fixed in XNU-8019.

More detail about identification and validation in the paper 😄

Thank you
Q&A